Títol: Millora de la gestió de recursos a SLURM

Volum: 1/1
Alumne: Carlos Fenoy García

Director/Ponent: Julita Corbalan González
Departament: Arquitectura de Computadors
Data: Juny 2010

DADES DEL PROJECTE

Títol del projecte:	Millora de la gestió de recursos a SLURM
Nom de l'estudiant:	Carlos Fenoy García
Titulació:	Enginyeria Superior en Informàtica
Crèdits:	37,5
Director/Ponent:	Julita Corbalan González
Departament:	Arquitectura de Computadors

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President:	Xavier Martorell Bofill
Vocal:	Anna de Mier Vinué
Secretari:	Julita Corbalan González

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Millora de la gestió de recursos a SLURM

Carlos Fenoy García

Juny 2010

Agraïments

Aquest projecte no hauria estat possible sense el recolzament i la col·laboració de moltes persones. M'agradaria dedicar-lo a totes elles.

A Julita Corbalan, que en tot moment m'ha ajudat i s'ha implicat molt en la realització d'aquest projecte.

Als excompanys de primera planta de l'equip de suport: Christian Simarro, Xavier Abellán, Jorge Rodríguez i Jorge Naranjo que sempre m'han "suportat" i m'han escoltat atentament.

A tot l'equip de sistemes i especialment a Javier Bartolomé, Jordi Valls, Alejandro Lucero i Juan Carlos Sánchez, que m'han donat "suport" i han aportat idees.

A David Vicente i Montserrat González, pel seu recolzament moral durant la realització d'aquest projecte.

A l'Ari per haver-se llegit tot el projecte i tot el suport moral que m'ha ofert durant tota la carrera i a la meva família, que a la seva manera sempre m'han recolzat.

En fi, a totes les persones que han suportat alguna de les meves ratllades i que sense elles aquest projecte no hauria estat possible.

Gràcies a tots.

Índex

Índex de figures

CAPÍTOL 1

Introducció

Aquest projecte pretén implementar i avaluar una millora en la selecció de recursos d'un gestor de recursos orientat a supercomputadors com MareNostrum, situat al BSC-CNS[1], basant-se en l'ús de recursos compartits.

1.1 Motivació

El món de la supercomputació avança molt ràpidament, s'ha passat de dos a dotze processadors per node i de prop de 5000 nodes a més de 25000 en els darrers 5 anys. Aquests augment en el nombre de recursos ha provocat que la gestió dels recursos sigui crítica per obtenir el màxim rendiment d'un supercomputador, i per maximitzar el nombre d'hores útils de càlcul. Si és té en compte que l'ús del supercomputador es determina per una assignació de determinades hores de càlcul a cada un dels usuaris de la màquina, es posa de manifest que la gestió de recursos no és tan sols important per la gestió del centre, sinó també pels usuaris, ja que una bona gestió de recursos provocarà que aquests puguin obtenir més rendiment de les seves hores assignades.

L'augment en la freqüència dels processadors ha provocat també que altres recursos resultin crítics durant l'execució dels programes. Un d'aquests recursos crítics és l'ample de banda de memòria, que augmenta d'una manera més pausada respecte la freqüència dels processadors, fet que provoca que aquest sigui un recurs realment crític, en esdevenir un dels principals colls d'ampolla d'algunes aplicacions. Per tant, una gestió de recursos conscient de l'ample de banda de memòria disponible en cada node, i dels requisits de cada aplicació, serà capaç de repartir les

[1] http://www.bsc.es

diferents tasques d'aquesta per maximitzar-ne el seu rendiment, reduir el seu temps d'execució i aconseguir que més aplicacions es puguin executar en un interval de temps.

L'existència de polítiques teòriques enfocades a pal·liar aquestes problemàtiques, conjuntament amb l'exposat anteriorment, són els motius que han impulsat la realització d'aquest projecte, per analitzar el comportament d'una política concreta, basada en la reducció de les penalitzacions inherents a la compartició de recursos, que no ha estat mai implementada en un entorn real.

1.2 Objectius

Donada la importància de la selecció de recursos per a l'òptim funcionament d'una màquina d'alt rendiment, l'objectiu serà implementar una política de selecció de recursos capaç de reduir la penalització provocada per la compartició de l'ample de banda de memòria entre diferents tasques dins d'un node de càlcul.

Les polítiques de selecció de recursos més utilitzades avui en dia només tenen en compte el número de recursos disponibles, els consideren consumibles que s'assignen a cada treball. Només es consideren recursos no compartits o recursos compartits sense permetre una assignació superior al disponible en un node. No es te en compte si l'assignació realitzada pot comportar alguna penalització per la compartició d'algun recurs del que no se'n tenen dades, com per exemple l'ample de banda de memòria o la xarxa de comunicacions.

Així doncs, l'objectiu d'aquest projecte és implementar un política de selecció de recursos que tingui en compte la disponibilitat d'un recurs compartit, com és l'ample de banda a memòria, i l'ús que en fan les aplicacions, i posteriorment analitzar-ne el seu comportament. D'aquesta manera es pretén obtenir una política de selecció de recursos ampliada amb noves funcionalitats sense renunciar a les què ja s'ofereixen actualment.

1.3 Metodologia

La metodologia seguida per a la realització d'aquest projecte ha estat la següent:

1. Estudi exhaustiu de la política Less Consume[2], per entendre completament el seu funcionament.

2. Estudi del gestor de recursos SLURM[3], del seu sistema de *plugins* i anàlisi de viabilitat de la implementació de la política Less Consume.

3. Identificació dels nous requisits dels treballs per tal de poder utilitzar la política, i implementació d'aquests: modificació de comandes per acceptar nous paràmetres i modificació

[2] Job-Guided Scheduling Strategies for Multi-Site HPC Infrastructures, [Gui]
[3] Simple Linux Utility for Resource Management, [SLU]

del controlador per a emmagatzemar-les i tenir-les en compte.

4. Implementació d'una primera versió de la política capaç d'assignar recursos segons l'ample de banda requerit per cada treball.

5. Implementació de millores a la política abans implementada, afegint opcions de configuració per modificar el comportament de la política segons les necessitats mitjançant paràmetres.

6. Modificació d'un *plugin* d'*accounting* encarregat d'enregistrar els events del treballs, que implementi un format estàndard d'*accounting* per poder analitzar el comportament de la política i els seus diferents paràmetres.

7. Preparació de l'entorn d'avaluació.

8. Realització de les proves i avalució dels resultats.

1.4 Organització de la memòria

Aquest document esta estructurat en els següents apartats:

- Introducció: En aquest capítol es fa un repàs dels objectius del projecte i la motivació. També es descriu mínimament la metodologia utilitzada per realitzar-lo.

- Entorn del projecte i estat de l'art: es dóna una introducció als conceptes que seran necessaris per entendre aquest projecte.

- Desenvolupament del projecte: en aquesta secció es descriu tot el desenvolupament dut a terme per a la implementació del projecte.

- Avaluació i anàlisi de resultats: aquí es presenta el desenvolupament previ per a la preparació de les proves, la preparació de l'entorn d'execució d'aquestes i l'anàlisi dels resultats.

- Estudi econòmic i planificació: aquest capítol dóna més detall sobre la planificaciói seguida en la realització del projecte així com el corresponent anàlisi econòmic del desenvolupament.

- Conclusions: Aquesta secció sintetitza els resultats obtinguts i extreu les conclusions que s'han extret d'aquests.

- Apèndix: finalment, es presenten en forma d'annexos els resultats de vàries de les proves realitzades.

CAPÍTOL 2

Entorn del projecte i estat de l'art

Oferim aquí una introducció a la supercomputació i als conceptes que apareixeran en aquest projecte. Així, es presenta una visió molt bàsica de què és un supercomputador, dels planificadors i gestors de recursos i de les polítiques de selecció de recursos.

2.1 Supercomputadors

Els supercomputadors són màquines pensades per a treballar amb problemes que amb un ordinador domèstic podrien trigar mesos o fins i tot anys a poder-se resoldre. Per fer-ho, la solució més habitual és connectar molts ordinadors entre sí, per a què treballin conjuntament i així oferir més recursos a una aplicació per a què s'executi més ràpidament o poder executar molts al mateix temps. Cada un dels ordinadors connectats rep el nom de node; normalment tenen més d'un processador i disposen d'una xarxa de gran velocitat i baixa latència per a les comunicacions entre les diferents tasques d'una aplicació.

Per poder controlar quines aplicacions s'executen en cada moment s'utilitzen sistemes de planificació - *batch systems*- que, ajudats pels gestors de recursos, s'encarreguen de decidir en quin moment i amb quins recursos s'executa cada treball. D'aquesta manera s'optimitza el percentatge d'utilització de la màquina. Per a poder-ho fer, però, cal que els usuaris donin una sèrie de requeriments dels seus treballs, com ara temps estimat d'execució, número de tasques, tasques per node, etc. D'aquesta manera el planificador té prou informació per exercir correctament la seva tasca.

MareNostrum

MareNostrum és el supercomputador que s'ha utlitzat per a realitzar les proves d'aquest projecte. Està situat al Barcelona Supercomputing Center - Centro Nacional de Supercomputación[BSC]. Disposa de 2560 nodes JS21; cada node -*blade*- consta de 2 processadors de 2 nuclis cada un, configurant un total de 10240 processadors, 8 GiB de memòria RAM i tres xarxes: una xarxa Myrinet per a les comunicacions de les aplicacions paral·leles, una xarxa Gigabit Ethernet per a l'accés al sistema de fitxers distribuïts i una xarxa Fast Ethernet per a l'administració i el manteniment del sistema.

En quant a software, els nodes utilitzen un sistema operatiu GNU/Linux SLES10SP2. El planificador de treballs utilitzat és Moab, que es coordina amb SLURM, que realitza, únicament, les tasques de gestor de recursos[1]. Aquesta configuració ofereix un rendiment màxim teòric de 94,21 TeraFlops, amb un rendiment sostingut de 63,83 TeraFlops segons Linpack, i una memòria total de 20TiB.

Respecte al seu ús per part dels usuaris, s'estableix un sistema de quotes per hores en base als projectes a desenvolupar al centre. Els diferents projectes es presenten a un comité independent d'experts, que en funció de les seves valoracions determinen l'assignació de recursos a cada un dels projectes.

2.2 Gestors de recursos

Els gestors de recursos són sistemes destinats a controlar en tot moment l'estat de totes les parts de la màquina. Informen al planificador de qualsevol incidència que pugui presentar un node i així poder-ho contemplar en planificar els treballs que té encuats. Podem trobar sistemes que incorporen el planificador i el gestor de recursos en una única aplicació, com SLURM, o sistemes en què planificador i gestor de recursos són aplicacions independents, com és el cas de la combinació Moab i SLURM, planificador i gestor respectivament. Amb aquesta configuració, Moab és el sistema responsable de la planificació, i cedeix la gestió dels treballs i les seves tasques al gestor de recursos amb el que es comunica, que passa a ser l'encarregat final de què els treballs s'executin on toca i amb el nombre de tasques sol·licitades per l'usuari.

Els planificadors i gestors de recursos necessiten disposar de diferents paràmetres per tal d'optimitzar l'assignació de recursos de cada treball. Aquests paràmetres són, bàsicament, duració -en hores- del treball, número de nodes que requereix i número de tasques per node.

SLURM

SLURM és un gestor de recursos desenvolupat al Lawrence Livermore National Laboratory (LLNL) i distribuit sota llicència GPL. Es basa en un sistema de *plugins*, el qual permet realitzar

[1] Com es veurà més endavant, SLURM pot dur a terme les dues tasques, això és, planificació i gestió de recursos, de forma simultània

modificacions a alguna de les parts de forma relativament senzilla sense perdre cap funcionalitat. Els *plugins* són habitualment petits programes amb una interfície comuna que permeten afegir funcionalitats a altres aplicacions. En aquest cas SLURM utilitza *plugins* pràcticament per a tot excepte la coordinació entre els diferents *plugins*. Actualment, disposa també de diverses polítiques de planificació, cada una implementada en un *plugin* diferent. Els diferents *plugins* d'SLURM són (en negreta els modificats en aquest projecte):

- *Accounting Storage*: *Plugin* responsable de comunicar-se amb una base de dades per obtenir la informació dels usuaris amb accés al sistema.
- *Authentication*: Autentica els serveis de cada node i evita intrusions.
- *Checkpoint*: Desa l'estat d'un treball cada cert temps, per a poder-lo recuperar en cas de fallada de hardware.
- *Cryptographic*: Estableix com es realitzaran les comunicacions entre els diferents daemons.
- *Job Accounting Gather*: Defineix quin sistema s'ha d'utilitzar, aix o linux, per recollir informació sobre els treballs.
- ***Job Completion***: Encarregat de mantenir un registre amb tots els esdeveniments referits als treballs que es produeixen.
- *MPI*: Defineix el tipus de xarxa *mpi* de què es disposa, per poder executar aplicacions paral·leles.
- *Priority*: Ens permet definir com es vol calcular la prioritat de cada treball.
- *Process Tracking*: Estableix com s'ha de determinar a qui pertany un procès concret.
- *Scheduling*: Permet definir la política de planificació.
- ***Resource Selection***: Controla la selecció de recursos disponibles.
- *Switch*: Defineix el tipus de xarxa de què es disposa.
- *Task*: Defineix si les tasques de les aplicacions has d'estar lligades a un processador en concret o no.
- *Topology*: Defineix la topologia de xarxa per optimitzar les comunicacions.

Com podem observar, SLURM disposa d'una bona quantitat de *plugins*, la qual cosa permet modificar gairebé qualsevol aspecte del seu funcionament per adaptar-lo a les nostres necessitats. SLURM utilitza tots aquests *plugins* per realitzar totes les tasques d'un planificador i d'un gestor de recursos.

Està dividit en un controlador (*slurmctld*) i un *daemon* a cada node (*slurmd*), que són els encarregats de comunicar al controlador quin és l'estat d'un node i de controlar l'estat del treballs en el seu node, El controlador, per la seva part, s'encarrega de processar totes les peticions dels usuaris, planificar els treballs i mantenir un registre del què succeix en tot moment. Així doncs, gairebé tota la feina la realitza el controlador, ja que els *slurmd* són programes dissenyats per a reduir al mínim la seva càrrega en cada node.

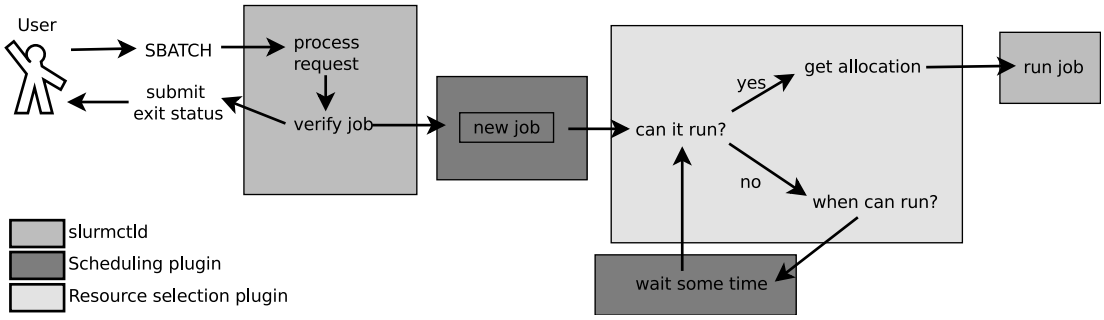

Figura 2.1: Diagrama de les fases d'un treball des de que l'usuari l'envia fins que comença a executar-se.

Les fases per les què passa un treball des de que un usuari realitza la petició d'encuar-lo fins que aquest comença a executar-se són, de manera molt simplificada, les següents:(fig.2.1)

Primer, un usuari envia un treball mitjançant la comanda *sbatch*. Aquesta es comunica amb l'*slurmctld*, que rep la petició d'un nou treball, la processa i comprova que la petició sigui correcta. Després, el nou treball passa al *plugin* de planificació que l'afegeix a la cua, i pregunta al *plugin* de selecció de recursos si es pot executar immediatament. En cas afirmatiu, demana una assignació de recursos, i sol·licita a *slurmctld* l'execució. Si no es pot executar en aquest moment, sol·licita una estimació al *plugin* de selecció de recursos i la desa. Cada vegada que finalitzi un treball o cada cert temps, el *plugin* de planificació comprovarà si el primer treball en cua es pot executar.

2.3 Less Consume

Less Consume és un política de selecció de recursos desenvolupada al BSC per en Francesc Guim, tema de recerca de la seva tesi doctoral, que no ha estat mai implementada en un entorn real. Està basada en la política FirstFit, que selecciona els recursos que permeten executar un treball el més aviat possible, afegint una nova restricció: la penalització resultant de l'ús d'ample de banda a memòria ha de ser el mínim possible, és a dir, el treball s'ha de veure afectat el mínim possible per compartir l'ample de banda amb cap altre treball o amb ell mateix. La penalització és el màxim de la penalització dels nodes de l'assignació, i aquesta és proporcional a la saturació de l'ample de banda de memòria. L'objectiu de la política és doncs, a partir d'una assignació inicial de referència, obtenir l'assignació amb la penalització mínima possible

sense augmentar el temps d'inici del treball.

La política utilitza un taula de reserva per mantenir en tot moment un registre de quins recursos s'han assignat a cadascun dels treballs. En aquest cas, els recursos són els processadors i el temps. Per tant, donada la taula de reserva, la descripció d'un treball (número de cpus, duració,...) i un temps d'inici (*treq*), Less Consume troba l'assignació que minimitza la penalització d'un treball. Per fer-ho segueix els següents passos:

1. Per cada processador de la taula de reserva es selecciona l'interval de temps més proper a *treq* que el satisfaci: el processador no està assignat durant aquest interval de temps i, la duració de l'interval és superior o igual a la duració requerida pel treball. Cadascun d'aquests intervals de temps associats a un processador(*Buckets*) s'afegeixen a una llista ordenada segons el seu temps d'inici.

2. D'aquest conjunt de *Buckets*, es seleccionen els n ($n = cpus$) primers que satisfacin que el seu interval de temps comparteixi almenys el temps requerit pel treball.

3. En cas que en el pas anterior el número de *Buckets* sigui inferior al número sol·licitat de cpus, implica que no hi havia suficients *buckets* que compartissin el temps necessari. En aquesta situació, es selecciona el primer *bucket* amb un temps d'inici (t_{i_0}) superior a *treq* i s'actualitza *treq* amb $treq = t_{i_0}$, i es tornen a iterar els passos 1, 2 i 3. En aquest punt, disposem de tres conjunts de *buckets*:

   - Els *buckets* descartats per què mai formaran part d'una assignació vàlida.
   - Els *buckets* seleccionats en aquest punt, i que satifan els requisits del treball. En cas que la penalització no es pugui disminuir amb una futura selecció, aquest serà el conjunt de *buckets* seleccionats per l'assignació.
   - Els *buckets* que encara no han estat evaluats. Aquests són els que s'utilitzaran per intentar millorar l'assignació inicial, intentant reduir al màxim la penalització.

4. Per cadascun dels *buckets* seleccionats, l'algoritme calcula la penalització associada en cas de què aquest *bucket* s'assignés al treball. Si tots els *buckets* tenen una penalització de 1, aquesta és l'assignació òptima i l'algorisme acabaria aqui.

5. Per cada *bucket* en el conjunt per evaluar:

   (a) Si el nombre de *buckets* al conjunt dels seleccionats és inferior als processadors sol·licitats, s'afegeix el *bucket* actual al conjunt de seleccionats i es continua amb la següent iteració.

   (b) De manera semblant al pas anterior, es calcula la penalització que provocaria la selecció d'aquest *bucket*.

(c) Es selecciona el *bucket* amb la màxima penalització del conjunt de *buckets* seleccionats (b_{Max}).

(d) En cas de què la penalització del *bucket* actual (b_{Act}) sigui inferior a la penalització del *bucket* b_{Max}, b_{Act} s'afegeix al conjunt de buckets seleccionats i es treu del conjunt de *buckets* per avaluar i b_{Max} s'afegeix al conjunt de *buckets* descartats i s'elimina del conjunt de *buckets* seleccionats. En aquest punt, el *bucket* afegit (b_{Act}) pot no compartir la finestra de temps amb els altres *buckets* del conjunt, aleshores:

i. Els *buckets* que no comparteixin la mateixa finestra de temps que b_{Act} es passen al conjunt de *buckets* descartats.

ii. Si el número de *buckets* del conjunt de seleccionats és el número de processadors requerits, s'estableix l'última assignació vàlida. Si la penalització de tots els *buckets* és inferior que la penalització mínima trobada fins al moment, el conjunt actual de *buckets* s'estableix com a assignació. Si no, l'algoritme continua amb la següent iteració per avaluar el següent *bucket* del conjunt per avaluar.

6. Es retorna l'assignaició amb la penalització mínima.

Less Consume Threshold

La política Less Consume, a diferència d'altres polítiques de selecció de recursos, millora el rendiment dels treballs que comparteixen recursos. La manera de fer-ho, però, té algunes desavantatges. La més important és que els treballs poden trigar molt més a executar-se, ja que per aconseguir una penalització mínima pot caldre esperar a que no hi hagi compartició de recursos. Aquest fet és el que intenta millorar la modificació de la política anomenada Less Consume Threshold, que mitjançant un nou paràmetre, el llindar, permet que els treballs puguin executar-se abans.

Per fer-ho, es configura un llindar. Aquest serà el punt a partir del cual una assignació es considerarà valida. D'aquesta manera no es busca l'assignació que proporcioni la menor penalització, sinó aquella que permeti executar el treball el més aviat possible amb una penalització per sota del llindar.

Les diferències principals entre les dues polítiques es troben en els punts 4) i 5.d.2), ja que en aquests casos si el conjunt de *buckets* seleccionats té una penalització inferior o igual al llindar, aquesta serà l'assignació final.

Aquesta política és més ràpida que Less Consume doncs una vegada té una assignació inferior al llindar establert acaba i no cerca més.

2.4 Standard Workload Format

L'*Standard Workload Format*[CCF+99] és un format proposat com a estandard per l'*Experimental Systems Lab* de la *The Hebrew University*. Permet desar tota la informació relativa a un treball en un format estandaritzat que es pot utilitzar posteriorment per analitzar el comportament de diversos component que influeixen en el comportament del treballs. Aquest format utilitza 18 característiques d'un treball, entre les què trobem: l'identificador, el temps d'execució, la mitjana d'ús dels processadors, l'usuari, etc. A la fig es pot observar un fragment d'un workload en el què es poden veure tots els camps inclosos en l'estandard.

Els camps inclosos en el format són:

Job Number, Submit Time, Wait Time, Run Time, Number of Allocated Processors, Average CPU Time Used, Used Memory, Requested Number of Processors, Requested Time, Requested Memory, Status, User ID, Group ID, Executable (Application) Number, Queue Number, Partition Number, Preceding Job Number i Think Time from Preceding Job.

```
9727   1847521   0    203   64  -1  -1  -1   -1  -1 -1  35  1 146  0 -1 -1 -1
9729   1847672   0     12    1  -1  -1  -1   -1  -1 -1   4  1   4  0 -1 -1 -1
9730   1847737   0    213   64  -1  -1  -1   -1  -1 -1  35  1 146  0 -1 -1 -1
9732   1848075   0   2650   32  -1  -1  -1   -1  -1 -1   4  1  -1  1 -1 -1 -1
9740   1849933   0     41    1  -1  -1  -1   -1  -1 -1  13  1  36  0 -1 -1 -1
9746   1850724   0   2657   32  -1  -1  -1   -1  -1 -1   4  1  -1  1 -1 -1 -1
9748   1850921   0     38    1  -1  -1  -1   -1  -1 -1  22  1 270  0 -1 -1 -1
9749   1851083   0     15   32  -1  -1  -1   -1  -1 -1  43  1 221  0 -1 -1 -1
```

Figura 2.2: Exemple del format Standard Workload Format.

CAPÍTOL 3

Desenvolupament del Projecte

3.1 Estudis previs

Al començament d'aquest projecte s'han estudiat la política de selecció de recursos Less Consume, detallada en la secció 2.3, i el funcionament general d'SLURM i del seu *plugin* de selecció de recursos i el *plugin* d'*accounting*.

Less Consume

S'ha estudiat la política Less Consume a partir de la tesi doctoral del Dr. Francesc Guim, atès que és la principal font d'informació d'aquesta política. S'ha decidit implementar i analitzar la versió Less Consume Threshold ja que ofereix algunes millores sobre la versió bàsica.

D'aquest estudi s'han extret el requisits principals per tal de poder implementar aquesta política.

SLURM

Per a realitzar l'estudi d'SLURM s'ha utilitzat la documentació disponible del projecte i, principalment, el seu codi font, sota llicència GPL. S'han estudiat els mecanismes de comunicació entre el programa per enviar treballs (*sbatch*) i el controlador, les estructures de dades que utilitza el controlador, el sistema de configuració d'SLURM i el funcionament i estructures del *plugin* de selecció de recursos.

D'aquests dos estudis s'extreuen els requisits preliminars per tal de poder implementar la política Less Consume dins l'estrucutra d'SLURM. Aquest requisits són:

- Addició d'un nou paràmetre de configuració de cada node, que contingui el valor de l'ample de banda de memòria (veure 3.2 Paràmetre de configuració).
- Addició de dos nous paràmetres per la descripció d'un treball, que continguin l'ample de banda de memòria requerit per un treball i la màxima penalització permesa per aquest.

3.2 Implementació dels requisits previs

3.2.1 Paràmetre de configuració

Per implementar el nou paràmetre de configuració de cada node, s'ha modificat l'estructura que emmagatzema la informació de cada node afegint un nou camp pel valor d'ample de banda de memòria. Un exemple de la configuració d'un node és:

```
NodeName=s04c1b13 Procs=4 State=IDLE MemoryBandwidth=3000
NodeName=s09c2b05 Procs=4 State=IDLE MemoryBandwidth=3000
NodeName=s09c2b11 Procs=4 State=IDLE MemoryBandwidth=3000
NodeName=s11c3b08 Procs=4 State=IDLE MemoryBandwidth=3000
```

On es pot apreciar que per cada node es configura el nom, el número de processadors, l'estat en què ha d'arrencar el node i el total d'ample de banda a memòria del què disposa, essent aquest últim el paràmetre afegit.

```
0  struct node_record {
     uint32_t magic;      /* magic cookie for data integrity */
     char *name;      /* name of the node. NULL==defunct */
     uint16_t node_state;     /* enum node_states, ORed with
              * NODE_STATE_NO_RESPOND if not
5             * responding */
     bool not_responding;     /* set if fails to respond,
              * clear after logging this */
     time_t last_response;    /* last response from the node */
     time_t last_idle;    /* time node last become idle */
10   uint16_t cpus;       /* count of processors on the node */
     uint16_t sockets;    /* number of sockets per node */
     uint16_t cores;      /* number of cores per CPU */
     uint16_t threads;    /* number of threads per core */
     uint32_t real_memory;    /* MB real memory on the node */
15   uint32_t mem_bandwidth;    /* MB real memory bandwidth on the node */
     uint32_t tmp_disk;     /* MB total disk in TMP_FS */
     struct config_record *config_ptr;  /* configuration spec ptr */
     uint16_t part_cnt;     /* number of associated partitions */
     struct part_record **part_pptr; /* array of pointers to partitions
```

```
20            * associated with this node*/
  char *comm_name;    /* communications path name to node */
  uint16_t port;      /* TCP port number of the slurmd */
  slurm_addr slurm_addr;    /* network address */
  uint16_t comp_job_cnt;    /* count of jobs completing on node */
25  uint16_t run_job_cnt;   /* count of jobs running on node */
  uint16_t no_share_job_cnt;  /* count of jobs running that will
            * not share nodes */
  char *reason;       /* why a node is DOWN or DRAINING */
  char *features;     /* associated features, used only
30            * for state save/restore, DO NOT
            * use for scheduling purposes */
  char *arch;     /* computer architecture */
  char *os;     /* operating system currently running */
  struct node_record *node_next;  /* next entry with same hash index */
35  uint32_t hilbert_integer; /* Hilbert number based on node name,
            * no need to save/restore */
#ifdef APBASIL_LOC
  uint32_t basil_node_id;   /* Cray/BASIL node ID,
            * no need to save/restore */
40 #endif  /* APBASIL_LOC */
};
```

Codi 3.1: Afegit el camp mem_bandwidth a l'estructura node_record (línia 15)

Per a poder reconèixer el nou paràmetre i emmagatzemar-lo a l'estructura descrita (codi 3.1), s'han modificat les funcions que s'encarreguen d'analitzar els arxius de configuració. Aquestes funcions són:

- *_parse_nodename* (common/read_config.c), encarregada d'omplir l'estructura *slurm_conf_node_t* de cada node amb les dades extretes de l'arxiu de configuració.
- *_build_all_nodeline_info*, omple l'estructura *config_record* per cada node (slurmctld/read_config.c).
- *create_node_record* (slurmctld/node_mgr.h), crea una estructura *node_record* i l'omple amb la informació de *config_record*.

3.2.2 Nous paràmetres de descripció d'un treball

Per tal de poder implementar la política Less Consume Threshold es necessita disposar de dos nous paràmetres en la descripció d'un treball. Per a implementar-los s'ha modificat *sbatch* i algunes estructures del controlador. La modificació d'*sbatch* permet especificar quin és l'ample de banda requerit per un treball i el llindar màxim permès de penalització. Tot i que a la política original el llindar de penalització màxima s'estableix de forma global, en aquest projecte s'ha

obtat per a implementar-lo per cada treball tot i que l'avaluació des realitzarà de forma general. Un exemple de l'ús d'*sbatch* és el següent:

```
sbatch -n2 -N2 --mem-bandwidth=2000 --lc-threshold=1.5 exec2.sh
```

En aquest exemple es defineix un treball que executarà l'*script exec2.sh* utilitzant dues tasques en dos nodes amb un ample de banda de 2000MB/s per tasca i una penalització màxima de 1,5. Per a què la comanda *sbatch* reconegui els dos nous paràmetres, s'ha modificat el mecanisme per a la lectura de paràmetres, l'estructura que els emmagatzema per després enviar-los al controlador i el mecanisme de comunicació entre *sbatch* i el controlador per tal d'enviar els nous paràmetres.

Per la part del controlador s'ha modificat l'estructura *job_record*, que conté tota la informació referent a un treball: el número de tasques, número de cpus, la prioritat, etc., afegint-hi tres nous camps:

- *mem_bandwidth*: emmagatzema el requisit d'ample de banda de memòria del treball.
- *lc_threshold*: emmagatzema el llindar màxim permès de penalització.
- *penalty*: emmagatzema la penalització del treball.

3.3 Implementació d'una primera versió

Per implementar la política Less Consume Threshold, s'ha decidit utilitzar com a punt de partida un *plugin* ja existent, i no crear-ne un des de zero. S'ha pres aquesta decisió per no perdre funcionalitats, que els *plugins* actuals ja ofereixen, i perquè afegir el control d'ample de banda en els *plugins* actuals era, a priori, l'opció més senzilla. SLURM defineix un conjunt de funcions que han d'aparèixer sempre en el *plugin*, i que seran les úniques que s'utilitzaran des de fora del mateix. La figura 3.1 mostra la llista de funcions més rellevants del *plugin* i marcades aquelles que s'han modificat.

La funció principal és *select_p_job_test*, encarregada de realitzar la selecció de recursos per a un treball o, depenent del mode en què es crida, de comprovar si el treball serà capaç d'executar-se en algun moment.

```
int select_p_job_test (struct job_record *job_ptr, bitstr_t *bitmap,
        int min_nodes, int max_nodes, int req_nodes, int mode);
```

Aquesta funció rep per paràmetre l'estructura que descriu el treball (*job_ptr*), un *bitmap* amb els possibles nodes a utilitzar, dels quals es seleccionaran el nodes a assignar(*bitmap*), el número mínim de nodes(*min_nodes*), el número màxim de nodes(*max_nodes*), el número de nodes sol·licitats pel treball(*req_nodes*) i el mode en què s'ha d'executar la funció(*mode*), que pot ser:

```
#Funcions de la API
_add_job_to_res
_rm_job_from_res
init
fini
select_p_node_init
select_p_job_test
#Funcions auxiliars
_can_job_run_on_node
_verify_node_state
_get_res_usage
_choose_nodes
_select_nodes
cr_job_test
_will_run_test
_dup_node_usage
#Funcions auxiliars afegides
_list_find_job_id_lc
recalculate_job_penalty
```

Figura 3.1: Llista de les funcions més importants del plugin de selecció de recursos

- SELECT_MODE_RUN_NOW: intenta planificar el treball en el moment present.
- SELECT_MODE_TEST_ONLY: prova si el treball podrà executar-se.
- SELECT_MODE_WILL_RUN: determina quan i on podrà executar-se el treball.

Aquesta funció, depenent amb quin mode hagi estat cridada, crida a *_will_run_test*, en el cas de què s'utilitzi el mode SELECT_MODE_WILL_RUN o a *cr_job_test* amb qualsevol dels altres dos modes.

La funció *_will_run_test* s'encarrega d'anar cridant a *cr_job_test*, i verificar si el treball es pot executar. Si no es pot executar, simula l'eliminació de l'assignació del primer treball en acabar i torna a comprovar si aleshores el treball es podrà executar. Es va repetint l'ultima operació fins que el treball es pot executar, i es marca com a temps estimat de començament la finalització de l'ultim treball eliminat.

La funció *cr_job_test* és la funció que realitza gairebé tota la feina de *select_p_job_test*. Segons es pot extreure de la documentació d'SLURM el procediment es pot dividir en tres passos:

- Comparar els nodes en el bitmap *avail* amb l'estat actual de cadascun d'aquests nodes, per tal de trobar els nodes disponibles que satisfacin els requisits del treball.
- Comprovar els nodes del bitmap *avail* amb recursos assignats.
- Seleccionar l'ús de recursos, dels restants al bitmap *avail*, per aquest treball, amb el posicionament influenciat per les assignacions existents.

I l'últim d'aquests en:

```
* Pas 1: Cerca nodes lliures en totes les particions. Si n'hi ha
*        dóna una assignació al treball i surt. Si no continua.
*
* Pas 2: Elimina els recursos que s'estan utilitzan actualment
*        per particions de major prioritat, i avalua si el treball
*        encara es pot executar. Si no surt.
*
* Pas 3: Cerca nodes lliures en les particions amb la mateixa
*         prioritat que la partició del treball. Si n'hi ha aleshores
*         ves al pas 6. Si no continua:
*
* Pas 4: Cerca una allocatació dins de la partició del treball.
*         Si no es troba cap assignació possible surt. Si no
*         continua:
*
* Pas 5: Assigna el treball i surt
*
* Pas 6: Assigna el treball i surt(Algoritme diferent del pas 5)
```

Per a realitzar tots aquests passos cr_job_test utilitza les funcions de la figura 3.2. De totes aquestes funcions, les que s'han modificat per a poder implementar la primera versió han estat:

- *cr_job_test* (veure explicació anterior)
- *_verify_node_state*, determina si un node es pot utilitzar pel treball que s'està avaluant.
- *_can_job_run_on_node*, determina quins recursos d'un node concret poden assignar-se al treball avaluat.

La funció *cr_job_test* s'ha modificat per a desar l'ample de banda de memòria que s'ha assignat al treball en cada node. Per fer-ho, s'ha afegit a l'estructura *select_job_res*, que defineix exactament quins recursos s'han assignat al treball,un camp per emmagatzemar l'ample de banda de memòria que s'ha assignat al treball a cada un dels node assignats per a la seva execució.

La funció *_verify_node_state* s'ha modificat afegint el control de l'ample de banda de memòria, per comprovar si el node en disposa de prou, per tal de descartar-lo en cas que no sigui així. Respecte el fragment de codi afegit veure Fig. 3.3.

La funció *_can_job_run_on_node* (veure Fig. 3.4) s'ha modificat per tenir en compte l'ample de banda de memòria en l'assignació de recursos. Així doncs, s'assignen els recursos segons

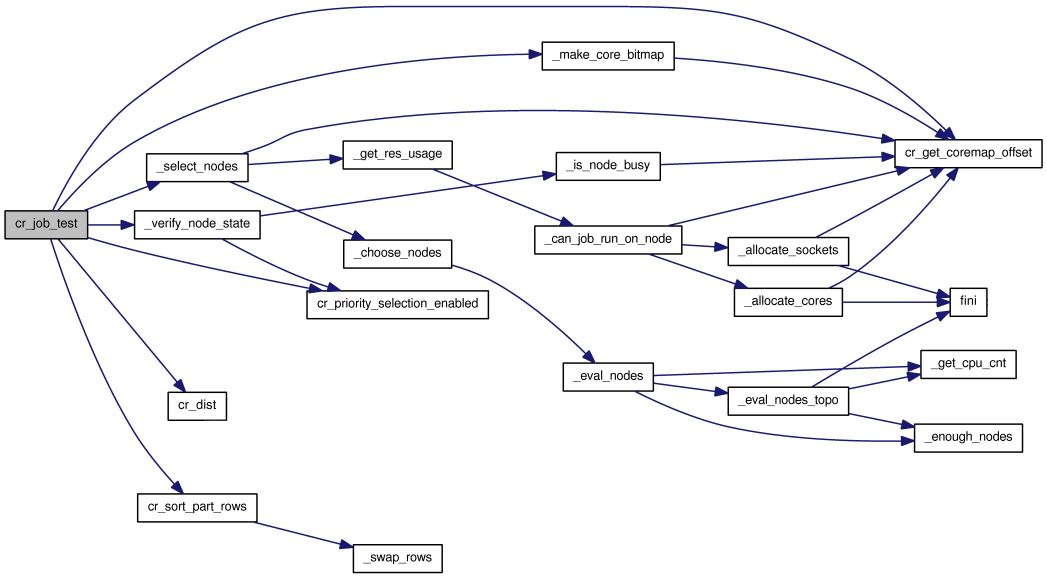

Figura 3.2: Diagrama de les fases d'un treball des de que l'usuari l'envia fins que comença a executar-se.

l'ample de banda disponible. Si un node disposa de 3000MB/s i un treball demana 1000MB/s, s'assignaran 3 tasques del treball a aquest node, ja que assignar-ne una més provocaria una sobrecàrrega dels recursos. Aquesta primera versió permetia assignar els recursos segons l'ample de banda de memòria, però sense tenir en compte si els treballs havien sol·licitat una penalització màxima (provocant una penalització màxima de 1 per aquest fet). Aquest fet comportava que no es poguéssin produir mai penalitzacions per la compartició de recursos, ja que no es podia assignar una cpu lliure d'un node ja ocupat per un treball si l'ús total d'ample de banda de memòria superava el disponible en el node.

3.4 Implementació de millores

Un cop avaluada i comprovat el seu correcte funcionament, aquesta primera versió es modifica per contemplar el paràmetre de llindar màxim de penalització i els paràmetres de configuració per modificar el comportament del *plugin*.

3.4.1 Paràmetre del llindar màxim de penalització

Per implementar el paràmetre del llindar màxim de penalització només s'ha modificat la funció *_can_job_run_on_node*; aquesta implementació, però, ha suposat la reestructuració completa del fragment exposat anteriorment(Veure Sec 3.4.2). Amb aquest nou paràmetre, la funció determina quantes cpus de cada node es poden assignar a un treball tenint en consideració

```
  /*
    * Check if there is enough memory bandwidth at the node
    */

   if (job_ptr->details->mem_bandwidth){
     free_bandwidth = select_node_record[i].mem_bandwidth;
     if (free_bandwidth*job_ptr->details->lc_threshold < min_bandwidth)
         {
       debug3("less_consume: _vns: node %s no bandwidth %u < %u",
         select_node_record[i].node_ptr->name,
         free_bandwidth, min_bandwidth);
       goto clear_bit;
     }
   }
```

Figura 3.3: Fragment de codi afegit a *_verify_node_state* per a controlar l'ample de banda de memòria disponible

```
req_mem_bandwidth = job_ptr->details->mem_bandwidth;
avail_mem_bandwidth = select_node_record[node_i].mem_bandwidth;
if (!test_only)
  avail_mem_bandwidth -= node_usage[node_i].alloc_mem_bandwidth;
/* memory bandwidth requested is per task */
while (cpus > 0  (req_mem_bandwidth * cpus) > avail_mem_bandwidth)
 cpus--;
if (cpus < job_ptr->details->ntasks_per_node){
  cpus = 0;
  bit_nclear(core_map, cr_get_coremap_offset(node_i),(
      cr_get_coremap_offset(node_i+1))-1);
}
```

Figura 3.4: Fragment de codi afegit a _can_job_run_on_node per a controlar l'ample de banda de memòria disponible

els possibles treballs existents en aquest node. Aquesta modificació, però, no és suficient atès que entre treballs no apareixerà mai penalització, ja que en la primera versió no es permet augmentar la penalització d'un treball ja existent en el node.

3.4.2 Nous paràmetres de configuració

Per aquest motiu, s'han afegit nous paràmetres de configuració. Aquests modifiquen el comportament del *plugin* per permetre l'augment de la penalització d'un treball fins al seu llindar i per augmentar el temps d'execució requerit en funció del llindar. Els possibles valors es recullen al paràmetre *SelectTypeParameters* i són:

LC_MEMORY , indica que el *plugin* ha de mantenir el seu comportament per defecte.

LC_PENALTY , que modifica el comportament permetent augmentar la penalització dels treballs existents en un node fins al seu llindar.

LC_REQ_TIME , que permet augmentar el temps d'execució requerit pel treball.

LC_PENALTY_REQ_TIME , combinació dels dos paràmetres anteriors.

Per a què aquests paràmetres de configuració estiguin disponibles, tan sols s'han hagut d'afegir a l'estructura *select_type_plugin_info* que descriu tots els possibles paràmetres dels *plugins* de selecció de recursos. Un cop s'ha disposat d'aquests nous paràmetres, s'ha modificat el codi per a tenir-los en compte, i modificar el comportament segons quin hagi estat seleccionat. Aquests canvis afecten principalment la funció *_can_job_run_on_node* (veure Fig 3.5) ja que, tal i com s'ha comentat prèviament, s'ha hagut de reestructurar tot el fragment vist anteriorment(veure Fig 3.4). D'aquesta manera, el codi queda dividit en dues parts segons el mode en què s'hagi cridat a la funció (*test_only*, normal). En el cas de *test_only*, mode en el què es comprova si un treball podrà executar-se mai en aquest node, només es té en compte el propi treball i no altres treballs que pugin estar executant-se en aquest moment en el node avaluat. En aquest mode no es pot saber quin serà l'estat del node un cop el treball comenci a executar-se. En el mode normal sí que es tenen en compte els treballs que puguin estar executant-se en el node, ja que en aquest mode es busca una assignació pel treball.

Primer es calcula l'ample de banda utilitzat en cas de fer servir tots els processadors lliures del node. Posteriorment es calcula la penalització i, si aquesta és superior al llindar establert, es va reduint el nombre de cpus assignades i recalculant la penalització fins que aquesta és inferior al llindar o fins que no queden cpus. Un cop establert el nombre final de cpus que es poden assignar, si l'ample de banda utilitzat és superior al disponible del node, es comprova si aquesta assignació afecta a l'execució dels altres treballs executant-se al node. En el cas de què hi hagi treballs executant-se, per cada un d'ells es comprova si s'està augmentant la seva penalització. Si és així, l'acció dependrà aleshores del paràmetre *SelectTypeParameters*; si està establert en un dels valors que permeten augmentar la penalització dels treballs existents, aleshores es

```
req_mem_bandwidth = job_ptr->mem_bandwidth;
avail_mem_bandwidth = select_node_record[node_i].mem_bandwidth;
threshold = job_ptr->lc_threshold;
if(test_only){
  alloc_mem_bandwidth = req_mem_bandwidth*cpus;
  new_penalty = (double)alloc_mem_bandwidth / avail_mem_bandwidth;
  while (cpus > 0  new_penalty > threshold ){
    cpus--;
    alloc_mem_bandwidth = req_mem_bandwidth*cpus;
    new_penalty = (double)alloc_mem_bandwidth / avail_mem_bandwidth;
  }
}else{
  alloc_mem_bandwidth = node_usage[node_i].alloc_mem_bandwidth +
      req_mem_bandwidth*cpus;
  /* memory requested is per task */
  alloc_mem_bandwidth = node_usage[node_i].alloc_mem_bandwidth +
      req_mem_bandwidth*cpus;
  new_penalty = (double)alloc_mem_bandwidth / avail_mem_bandwidth;
  while (cpus > 0  new_penalty > threshold ){
    cpus--;
    alloc_mem_bandwidth = node_usage[node_i].alloc_mem_bandwidth +
        req_mem_bandwidth*cpus;
    new_penalty = (double)alloc_mem_bandwidth / avail_mem_bandwidth;
  }
  alloc_mem_bandwidth = node_usage[node_i].alloc_mem_bandwidth +
      req_mem_bandwidth*cpus;
  if(alloc_mem_bandwidth > avail_mem_bandwidth){
    new_penalty = alloc_mem_bandwidth / avail_mem_bandwidth;
    job_iterator = list_iterator_create(node_usage[node_i].job_list);
    while(node_job_ptr = (struct job_record *) list_next(job_iterator)){
      if( cr_type == LC_PENALTY || cr_type == LC_REQ_TIME_PENALTY ){
        if( new_penalty > node_job_ptr->lc_threshold ) cpus = 0;
      }else if(new_penalty > node_job_ptr->penalty) cpus = 0;
    }
  }
}
```

Figura 3.5: Modificació de la funció _can_job_run_on_node per a tenir en compte els nous paràmetres

```
struct node_use_record {
        uint16_t node_state;            /* see node_cr_state comments */
        uint32_t alloc_memory;          /* real memory reserved by
            already
                                         * scheduled jobs */
        uint32_t alloc_mem_bandwidth;   /* allocated memory bandwidth */
        List job_list;                  /* list of running jobs */
};
```

Figura 3.6: Estructura node_use_record amb els dos nous camps

comprova que la nova penalització no sigui superior al llindar de cada un dels treballs. Si és superior no es pot assignar cap cpu. Si, en canvi, el paràmetre no ens permet augmentar la penalització, es comprova que la penalització dels treballs sigui menor a l'obtinguda en aquest node, i si no és així tampoc s'assigna cap cpu.

Per poder conèixer quins són els treballs existents en un node, s'ha afegit una llista dels treballs executant-se en un node, a l'estructura *node_use_record*(veure Fig. 3.6), que conté la informació dels recursos assignats d'un node. Un treball s'afegeix a aquesta llista, dins de la funció *_add_job_to_res*. Aquesta és la funció que s'executa just abans que un treball comenci el procés d'execució, i per tant és també aquí on s'actualitzen els valors de l'estructura *node_use_record* amb el nou consum, i es recalcula la penalització de cada un dels treballs amb els què el treball apunt de començar compartirà node. En el cas que s'hagi seleccionat un paràmetre del *plugin* que permeti augmentar el temps requerit d'un treball, també és en aquesta funció on s'incrementa, i s'incrementa segons el llindar establert pel treball, ja que, encara que en el moment de començar a executar-se no tingui penalització, és possible que en un futur la tingui, i per tant el més segur és augmentar la penalització just en el moment de l'inici de l'execució. De la mateixa manera que s'afegeix un treball a la llista de treballs d'un node, també s'elimina d'aquesta llista quan el treball acaba. Aquesta tasca es du a terme a la funció *_rm_job_from_res*, que de la mateixa manera que *_add_job_to_res*, també actualitza els valors de *node_use_record*, però en aquest cas reduint els recursos segons els consum del treball que s'elimina.

3.5 Modificació del plugin d'accounting

El *plugin* d'*accounting Job Completion*, és l'encarregat d'escriure en un arxiu las dades d'un treball un cop aquest finalitza. El format per defecte, però, no és suficient per a poder obtenir una informació detallada de l'execució d'un treball, doncs no ofereix informació sobre l'ample de banda utilitzat pel treball ni en quins processadors s'ha executat.

```
JobId=21065 UserId=fenoy(100) GroupId=Users(100) Name=test JobState=COMPLETED
Partition=projects TimeLimit=22 StartTime=2009-06-04T18:36:14
EndTime=2009-06-04T18:36:24 NodeList=cabezatest NodeCnt=1 ProcCnt=1 WorkDir=/home/user
```

Per tal de poder analitzar el comportament del *plugin* de selecció de recursos es necessita

informació més detallada, que descrigui apart de la informació vista anteriorment, l'ample de banda de memòria, la penalització, el llindar i el temps d'enviament del treball. Per a fer-ho s'ha modificat el *plugin* de *Job Completion*, per què mostri tota aquesta informació en un format estàndard, tal i com s'ha vist anteriorment(Secció 2.4). Aquest format, però, no inclou la informació exclusiva de la política Less Consume, com l'ample de banda requerit per un treball, o la penalització a la que aquest s'ha vist sotmès.

La informació que s'ha afegit al format és:

- ample de banda de memòria requerit per un treball.
- llindar màxim de penalització permès durant la execució.
- penalització màxima a la que el treball s'ha vist sotmès.
- Temps requerit original.
- Un número indicant l'assignació de processadors.
- Una cadena de zeros i uns indicant l'assignació de processadors.

Així doncs, el format final queda com l'original amb els sis nous camps afegits al final.

```
35 1275411912 31576 1273 8 -1 -1 8 24 0 COMPLETED bsc99368 bsc99 script\_34.cmd
-1 -1 -1 -1 1660 1.100000 1.000000 22 286331153 00010001000100010001000100010001
```

CAPÍTOL 4

Avaluació i anàlisi de resultats

Un cop descrit el desenvolupament del projecte, aquest capítol es centra en la preparació i execució de les proves i en l'anàlisi dels seus resultats.

4.1 Entorns de treball

Per cada fase del projecte s'ha utilitzat un entorn de proves i de treball diferent, segons les necessitats. S'exposa a continuació una breu ressenya dels entorns de treball utilitzats, la seva descripció i per a què s'ha utilitzat.

4.1.1 Desenvolupament: Ordinador personal

Durant l'etapa de desenvolupament del projecte s'ha utilitzat un ordinador personal. Es tracta d'un ordinador de sobretaula amb un processador AMD X2 amb doble nucli. Aquest ha estat la base per a tot el desenvolupament i el que s'ha utilitzat com a punt de connexió a les diverses màquines on s'ha realitzat l'anàlisi. Aquest ordinador ha estat, també, la plataforma de desenvolupament dels dos *plugins* d'SLURM, i de la validació del correcte funcionament. Per realitzar aquesta validació, s'ha instal·lat SLURM amb el paràmetre de configuració *–enable-multiple-slurmd*, que permet executar varis *daemons slurmd*, per simular un clúster més gran que el propi ordinador. D'aquesta manera, s'han configurat 5 nodes, per poder comprovar tant el funcionament correcte del *plugin* de selecció de recursos com del *plugin* d'*accounting*. Del primer s'ha pogut comprovar l'assignació correcta dels recursos segons l'ample de banda de memòria disponible, mentre que del segon s'ha comprovat, apart de què aparegués tota la informació

necessària, que la cadena que indica els processadors assignats apareixia correctament. Així i tot, un ordinador personal emulant 5 nodes s'ha mostrat unsuficient per a realitzar les proves que validin el correcte funcionament del *plugin*, doncs tot i l'emulació, realment només es disposa d'un node, i els recursos d'aquest es reparteixen en els 5 configurats.

4.1.2 Tests previs: Clustertest

Atès que el BSC-CNS ha permès la utilització per a les proves d'aquest projecte d'un clúster format per 3 nodes JS20, amb 2 processadors, s'han realitzat les proves de rendiment d'aplicacions en aquest sistema. No ha calgut instal·lar el *plugin* desenvolupat en aquest projecte, doncs el clúster ja disposa d'un sistema de cues, que és suficient per a les proves realitzades. En vistes dels resultats de les proves efectuades, però, s'ha descartat l'ús d'aquest clúster, doncs en disposar només de dos processadors per node, és difícil trobar aplicacions que saturin l'ample de banda, i no permet mostrar el correcte funcionament del *plugin*, ni les diferències de la política implementada en aquest projecte respecte a d'altres polítiques ja existents.

4.1.3 Avaluació: MareNostrum

Per a l'avaluació final s'ha utilitzat el supercomputador MareNostrum, descrit a la secció 2.1. Aquest supercomputador resulta adient per la realització de les proves en disposar de 4 processadors per node, la qual cosa permetrà saturar l'ample de banda de memòria, i observar com es comporten les aplicacions segons les diferents polítiques de selecció de recursos.

4.2 Preparació de l'entorn d'avaluació

4.2.1 Preparació de l'entrorn d'execució d'SLURM

Per a poder executar les proves s'han de complir uns requisits previs. Es volen executar les proves en paral·lel pel què es necessita una configuració d'SLURM dinàmica i que tots els arxius que utilitzi estiguin en un directori segons l'execució. També es vol poder consultar quin és l'estat de cada prova des dels nodes interactius de MareNostrum, doncs els usuaris no poden accedir als nodes de càlcul. Per a poder utilitzar SLURM s'ha hagut d'encapsular l'execució d'aquest dins d'un treball, doncs MareNostrum utilitza SLURM per a la seva gestió de recursos. S'ha modificat el fitxer de configuració per a què permeti una **configuració dinàmica dels nodes** (Veure Fig. 4.1), doncs a priori no es pot saber a quins nodes anirà a parar l' execució del treball.

En aquest arxiu de configuració es poden veure els paràmetres més importants per a l'execució d'SLURM d'una manera dinàmica. Per una part, al principi de tot de l'arxiu, s'estableix el **node de control**, on s'executa l'*slurmctld*, mitjançant la paraula clau *NODE1*. A continu-

```
#SLURM Controller
ControlMachine={NODE1}
 ...
#Ports a utilitzar i arxius d'identificacio del proces
SlurmctldPidFile={WORKDIR}/var/run/slurmctld.pid
SlurmctldPort=7001
SlurmdPidFile={WORKDIR}/var/run/slurmd.%n.pid
SlurmdPort=7009
 ...
# Plugin de seleccio de recursos
SelectType=select/lessc
SelectTypeParameters=LC_REQ_TIME_PENALTY
 ...
#Plugin d'accounting
JobCompType=jobcomp/fileswf
JobCompLoc={WORKDIR}/acct/job_comp

#Configuracio dels Logs
SlurmctldDebug=3
SlurmctldLogFile={WORKDIR}/logs/slurmctld.log
SlurmdDebug=3
SlurmdLogFile={WORKDIR}/logs/slurmd.%n.log
 ...
# COMPUTE NODES
NodeName={NODE2} Procs=4 State=IDLE MemoryBandwidth=3000 Port=7009
NodeName={NODE3} Procs=4 State=IDLE MemoryBandwidth=3000 Port=7009
NodeName={NODE4} Procs=4 State=IDLE MemoryBandwidth=3000 Port=7009
NodeName={NODE5} Procs=4 State=IDLE MemoryBandwidth=3000 Port=7009
NodeName={NODE6} Procs=4 State=IDLE MemoryBandwidth=3000 Port=7009
NodeName={NODE7} Procs=4 State=IDLE MemoryBandwidth=3000 Port=7009
NodeName={NODE8} Procs=4 State=IDLE MemoryBandwidth=3000 Port=7009
NodeName={NODE9} Procs=4 State=IDLE MemoryBandwidth=3000 Port=7009
PartitionName=debug Nodes={NODE2} Default=YES MaxTime=INFINITE State=UP
PartitionName=debug Nodes={NODE3} Default=YES MaxTime=INFINITE State=UP
PartitionName=debug Nodes={NODE4} Default=YES MaxTime=INFINITE State=UP
PartitionName=debug Nodes={NODE5} Default=YES MaxTime=INFINITE State=UP
PartitionName=debug Nodes={NODE6} Default=YES MaxTime=INFINITE State=UP
PartitionName=debug Nodes={NODE7} Default=YES MaxTime=INFINITE State=UP
PartitionName=debug Nodes={NODE8} Default=YES MaxTime=INFINITE State=UP
PartitionName=debug Nodes={NODE9} Default=YES MaxTime=INFINITE State=UP
```

Figura 4.1: Fragments més rellevants de l'arxiu de configuració d'SLURM

ació es configuren els **ports** que s'utilitzaran per a les comunicacions entre el controlador i **els nodes**, i on es desarà l'identificador de procés de cada *daemon*.

Seguidament es configura quin *plugin* de selecció de recursos s'utilitzarà. En aquest cas està configurat el *plugin lessc* (Less Consume) amb el paràmetre LC_REQ_TIME_PENALTY. En una altra secció es configura el *plugin* d'*accounting*, en aquest cas *fileswf* (Standard Workload Format), on es desarà l'arxiu d'*accounting*, els nivells de *log* que utilitzaran els diferents *daemons*, i on es desarà aquest log. I per últim, la configuració dels nodes. Una línia per cada node amb una paraula clau *NODEn* i una línia per afegir-lo a una partició.

Mitjançant un *script*(Veure Fig. 4.2), es substitueixen unes paraules clau(*NODEn*, *WORKDIR*), pel nom dels nodes assignats i el directori de treball, es creen els directoris necessaris

per a l'execució d'SLURM i es crea un arxiu per a poder carregar la configuració d'aquest SLURM encapsulat per poder veure la cua de treballs des d'un dels nodes interactius de MareNostrum. D'aquesta manera si es carrega l'arxiu generat(*sconf.sh*) executant la comanda d'SLURM *squeue* es pot veure quin és l'estat de la cua de treballs en un moment donat(Veure Fig. 4.3).

L'*script submitter.pl* és l'encarregat d'anar executant el workload, és a dir, d'anar enviant els treballs en el moment correcte.

4.2.2 Determinació de l'ample de banda de memòria disponible

Per tal de determinar l'ample de banda de memòria per posar-lo a la configuració d'SLURM, es podria utilitzar l'ample de banda de memòria nominal, o sigui, el que proporciona el fabricant. Aquest, però, està lluny de l'ample de banda real el qual es pot arribar a assolir durant l'execució d'una aplicació. Per tal de determinar doncs l'ample de banda, s'ha utilitzat un software anomenat *STREAM*[1] *(Sustainable Memory Bandwidth in High Performance Computers)*, que realitza unes proves de rendiment i ofereix una mesura de l'ample de banda de memòria molt més ajustada a la realitat.

El resultat de l'execució d'STREAM(Veure Fig. 4.4) dóna uns valors d'ample de banda entre 3033MB/s i 3601MB/s. Per no perjudicar l'execució de les aplicacions, s'ha decidit configurar els nodes amb un valor de 3000MB/s. D'aquesta manera sempre hi haurà un petit marge, independentment de quin tipus d'operacions realitzin les aplicacions, i no es penalitzaran per haver escollit un limit massa alt en cas que diverses tasques comparteixin un mateix node.

4.3 Preparació de les proves

Per analitzar el comportament del *plugin* de selecció de recursos s'ha decidit utilitzar els benchmarks NAS-PB (NASA Advanced Supercomputing Parallel Benchmarks), que són un conjunt de programes per analitzar diversos paràmetres de rendiment d'un sistema i permeten executar-los en diferents configuracions tant de càrrega de treball com de cpus a utilitzar. Els NAS-PB disposen de vàries configuracions de càrrega anomenades CLASS. Aquestes van des de la A fins a la D, de menys a més càrrega de treball respectivament, i la majoria de tests es poden configurar per utilitzar diverses cpus. Per tal de determinar quins tests dels inclosos en NAS-PB són més convenients d'executar per realitzar les proves, s'han extret traces de l'execució d'aquestes amb l'eina *mpitrace*, la qual ens permet obtenir diversos paràmetres de l'execució d'una aplicació. Un cop obtingudes les traces, l'aplicació *Paramedir* proporciona la informació desitjada en un format força clar. En aquest cas l'estudi s'ha centrat en el paràmetre d'ample de banda de memòria. S'han realitzat execucions de tots els tests, utilitzant diverses configuracions de *class* i nombre de cpus. Així i tot, però, cap dels tests anteriors assoleix un nivell prou

[1] http://www.cs.virginia.edu/stream/

```
#!/bin/bash
# @ job_name   = test_slurm
# @ initialdir = .
# @ output   = test1_%j.out
# @ error    = test1_%j.err
# @ total_tasks = 9
# @ cpus_per_task = 4
# @ wall_clock_limit = 10:00:00
# @ tracing = 1

export WORKDIR=$PWD;
mkdir -p var/run;
mkdir etc;
mkdir logs;
mkdir acct;
mkdir tmp;
cp /home/bsc99/bsc99368/slurm/etc/slurm.key $WORKDIR/etc/slurm.key
cp /home/bsc99/bsc99368/slurm/etc/slurm.cert $WORKDIR/etc/slurm.cert

j=1;
i=1;
for j in `sl_get_machine_list`;do
array[$i]=$j;
((j++));
((i++));
done;

sed -e s/{NODE1}/${array[1]}/ \
    -e s/{NODE2}/${array[2]}/ \
    -e s/{NODE3}/${array[3]}/ \
    -e s/{NODE4}/${array[4]}/ \
    -e s/{NODE5}/${array[5]}/ \
    -e s/{NODE6}/${array[6]}/ \
    -e s/{NODE7}/${array[7]}/ \
    -e s/{NODE8}/${array[8]}/ \
    -e s/{NODE9}/${array[9]}/ \
    -e s:{WORKDIR}:${WORKDIR}: $WORKDIR/slurm.conf.orig > $WORKDIR/etc/slurm.
        conf;

srun ./slurmd.sh
sleep 30;
export SLURM_CONF="$WORKDIR/etc/slurm.conf";
~/slurm/sbin/slurmctld -vvvv -c -D ;
sleep 10

export PATH=$PATH:/home/bsc99/bsc99368/slurm/bin/
echo "export SLURM_CONF=$SLURM_CONF" > sconf.sh

#Script encarregat d'executar el workload
./submitter.pl
wait;
```

Figura 4.2: Script per a llançar SLURM com a un treball

```
JOBID PARTITION                 NAME    STATE       TIME TIMELIMIT  NODES CPUS NODELIST(REASON)
    8     debug   workload_script_7.cmd  PENDING       0:00      2:00      1 4 (Resources)
   16     debug  workload_script_15.cmd  PENDING       0:00     38:00      1 4 (Priority)
   17     debug  workload_script_16.cmd  PENDING       0:00 UNLIMITED      2 8 (Priority)
   19     debug  workload_script_18.cmd  PENDING       0:00     22:00      4 16 (Priority)
   23     debug  workload_script_22.cmd  PENDING       0:00     10:00      4 16 (Priority)
   28     debug  workload_script_27.cmd  PENDING       0:00   1:28:00      1 4 (Priority)
   31     debug  workload_script_30.cmd  PENDING       0:00      3:00      2 8 (Priority)
   32     debug  workload_script_31.cmd  PENDING       0:00      5:00      4 16 (Priority)
   35     debug  workload_script_34.cmd  PENDING       0:00     22:00      2 8 (Priority)
   43     debug  workload_script_42.cmd  PENDING       0:00 UNLIMITED      2 8 (Priority)
   46     debug  workload_script_45.cmd  PENDING       0:00      5:00      2 8 (Priority)
   48     debug  workload_script_47.cmd  PENDING       0:00   1:14:00      2 8 (Priority)
   50     debug  workload_script_49.cmd  PENDING       0:00 UNLIMITED      4 16 (Priority)
   51     debug  workload_script_50.cmd  PENDING       0:00      7:00      1 4 (Priority)
   27     debug  workload_script_26.cmd  RUNNING      11:48   1:26:00      3 8 s23c3b[13-14],s23c4b01
   30     debug  workload_script_29.cmd  RUNNING      50:36   1:33:00      3 4 s23c3b[11-13]
   38     debug  workload_script_37.cmd  RUNNING      38:31   2:05:00      1 1 s23c3b12
   49     debug  workload_script_48.cmd  RUNNING      36:14     41:00      4 4 s23c3b14,s23c4b[01-03]
```

Figura 4.3: Estat de la cua de treballs

```
-------------------------------------------------------------
STREAM version $Revision: 5.9 $
-------------------------------------------------------------
This system uses 8 bytes per DOUBLE PRECISION word.
-------------------------------------------------------------
Array size = 20000000, Offset = 0
Total memory required = 457.8 MB.
Each test is run 20 times, but only
the *best* time for each is used.
-------------------------------------------------------------
Printing one line per active thread....
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 125990 microseconds.
   (= 125990 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:         3601.8261       0.1193       0.0888       0.1571
Scale:        3398.9411       0.1191       0.0941       0.1480
Add:          3304.4663       0.1779       0.1453       0.2037
Triad:        3033.3838       0.2170       0.1582       0.2821
-------------------------------------------------------------
Solution Validates
-------------------------------------------------------------
```

Figura 4.4: Resultats execució d'STREAM

```
; Version: 2
; Acknowledge: Uri Lublin, Hebrew University
; Information: http://www.cs.huji.ac.il/labs/parallel/workload
; MaxJobs: 50
; MaxRecords: 50
; MaxNodes: 32
; MaxRuntime: 8103
    1      18 -1       2   5 -1 -1 -1 -1 -1 1 -1 -1 -1 0 -1 -1 -1
    2      46 -1      11   2 -1 -1 -1 -1 -1 1 -1 -1 -1 0 -1 -1 -1
    3     233 -1      15   4 -1 -1 -1 -1 -1 1 -1 -1 -1 0 -1 -1 -1
    4     271 -1    4975   1 -1 -1 -1 -1 -1 1 -1 -1 -1 0 -1 -1 -1
    5     322 -1       4   1 -1 -1 -1 -1 -1 1 -1 -1 -1 0 -1 -1 -1
    6     369 -1       7   2 -1 -1 -1 -1 -1 1 -1 -1 -1 0 -1 -1 -1
    7     573 -1       4   4 -1 -1 -1 -1 -1 1 -1 -1 -1 0 -1 -1 -1
    8     620 -1      13   4 -1 -1 -1 -1 -1 1 -1 -1 -1 0 -1 -1 -1
    9     668 -1       7   4 -1 -1 -1 -1 -1 1 -1 -1 -1 0 -1 -1 -1
   10     855 -1      37   8 -1 -1 -1 -1 -1 1 -1 -1 -1 0 -1 -1 -1
```

Figura 4.5: Part del *workload* generat a partir del model de Uri Lublin i Dror G. Feitelson

alt d'utilització de l'ample de banda de memòria, i per això s'ha desenvolupat un *benchmark* propi(veure Apèndix B), altament configurable i que s'ha anomenat *sintètic*, per realitzar el màxim d'operacions amb memòria possibles i així aconseguir uns nivells més alts d'utilització de l'ample de banda de memòria. Amb aquest *benchmark* s'ha assolit un ample de banda mig durant l'execució de 1935MB/s per cada procés, mentre que el test que més consum té del conjunt de NAS-PB arriba a 1660MB/s.

4.4 Preparació dels workloads

Normalment els *workloads* que s'utilitzen per a comprovar el funcionament de les polítiques de planificació o de selecció de recursos provenen de dues fonts. La primera són els repositoris de *worklods* que contenen els registres d'alguns sistemes de varis mesos d'execucions reals. L'altra font poden ser els models que generen *workloads* segons les necessitats.

Per generar els *workloads*, conjunt d'aplicacions que s'executaran per avaluar el *plugin* de selecció de recursos, s'ha decidit utilitzar un model dissenyat per Uri Lublin i Dror G. Feitelson[LF03]. Aquest model disposa d'un codi capaç de generar un conjunt de treballs a partir d'unes especificacions, el qual s'ha utilitzat per a generar una llista de 50 treballs amb un màxim de 32 nodes i un temps màxim d'execució de 8103 segons(Veure Fig. 4.5). Atès que aquest model no és capaç de generar el número de tasques de cada aplicació, sinó que genera el nombre de nodes utilitzats, s'ha considerat el nombre de nodes com si fóssin cpus. Un cop obtingut el *workload* base, s'ha de generar la seqüència de treballs segons aquest *workload*. S'ha creat un script en llenguatge perl que a partir del *workload* base, d'un arxiu amb les aplicacions a utilitzar i d'uns límits segons el percentatge d'aplicacions amb una alt, mig o baix consum d'ample de banda de memòria, genera un script que podrem executar per a llançar tots els treballs a cues seguint les especificacions dels *workloads*. A l'arxiu d'aplicacions es troben els paràmetres característics per a la configuració de cada treball. Conté doncs, el tipus d'aplicació

```
#type   tasks   niters  time    mem_bandwidth   appname
h       8       10      1429    1600            cg.D.8
h       8       100     12800   1660            cg.D.8
h       16      10      411     1500            cg.D.16
h       32      10      251     1440            cg.D.32
h       1       60      1200    1935            sintetichigh
h       2       60      1200    1935            sintetichigh
h       2       2       41      1935            sintetichigh
h       4       60      1200    1935            sintetichigh
m       4       40      132     800             cg.C.4
m       16      10      411     800             cg.C.16
m       1       60      1200    1050            sinteticmed
m       2       60      1200    1050            sinteticmed
m       2       2       41      1050            sinteticmed
m       4       60      1200    1050            sinteticmed
l       2       200     328     570             cg.B.2
l       4       200     200     580             cg.B.4
l       8       200     107     445             cg.B.8
l       16      400     346     480             sp.C.16
l       1       60      1200    83              sinteticlow
l       2       60      1200    83              sinteticlow
l       8       2       41      83              sinteticlow
l       4       60      1200    83              sinteticlow
```

Figura 4.6: Arxiu d'aplicacions utilitzat per a la generació dels *workloads*

que és:

- h: Alt consum d'ample de banda de memòria $>$ 1400MB/s
- m: Mig consum d'ample de banda de memòria $\simeq$ 900MB/s
- l: Baix consum d'ample de banda de memòria $<$ 600MB/s

També conté el número de tasques de l'aplicació, el número d'iteracions amb les que s'ha executat, el temps que ha durat l'execució, l'ample de banda de memòria que ha consumit i el nom de l'aplicació. Amb aquests paràmetres i unes petites fòrmules per calcular el nombre d'iteracions que cal executar, s'assigna una aplicació a cada un dels treballs descrits al *workload* base, intentant mantenir al màxim el temps marcat. La sortida de l'*script* és, per una part, un resum del *workload* (Veure Fig 4.7) i per un altra l'*script* que s'ha de desar en un fitxer(submitter.pl) per a poder-lo executar un cop s'ha iniciat SLURM(Veure Fig. 4.8).

La figura 4.7 mostra les primeres línies del resum del *workload*. Les primeres línies corresponen a l'encapçalament i descriuen amb quins paràmetres s'ha executat l'*script* que genera els *workloads*. Les files *HIGH*, *MEDIUM* i *LOW* corresponen al nombre de treballs que hi ha de cada tipus. A partir d'aquí comença la descripció del treballs. Per cada un d'ells s'observa el temps en què s'ha de llançar el treball, el temps sol·licitat d'execució, el nombre de tasques, l'aplicació que executarà, el nombre d'iteracions que realitzarà i l'ample de banda que consumeix aquesta aplicació.

Pel què fa a l'script *submitter.pl*(Veure Fig. 4.8, cada treball consta de dues línies. La

```
High:   80
Medium: 10
Low:    10
File:   workload.txt
HIGH: 44
MEDIUM: 3
LOW: 3
   18    43    43   4          mg.C.4     3    88
   46   267   267   2    sintetichigh    16  1935
  233   368   368   4    sintetichigh    22  1935
  271  4862  4862   8          cg.D.8    34  1600
  322    83    83   1    sintetichigh     5  1935
  369   167   167   2    sintetichigh    10  1935
  573    83    83   4    sintetichigh     5  1935
  620   323   323   4          cg.C.4    98   800
  668   173   173   4          mg.C.4    12    88
  855   924   924   8          mg.C.8   528   440
```

Figura 4.7: Fragment del resum del workload generat

primera correspon a l'espera per a llançar el treball. El *workload* base defineix en quin moment s'ha d'enviar un treball a la cua, i per tant aquí s'ha de realitzar una espera abans de poder-lo enviar. La segona línia correspon a l'execució de la comanda *sbatch* amb tots el paràmetres necessaris per a la correcta execució de l'aplicació. Els paràmetres que s'utilitzen són:

- `-n`: nombre de tasques
- `--time`: temps requerit d'execució
- `--mem-bandwidth`: ample de banda de memòria que requereix aquesta aplicació
- `--lc-threshold`: llindar màxim permès de penalització

La comanda *sbatch* no pot executar una aplicació, sinó que ha d'executar un script. És per això que l'ultim paràmetre que apareix a la línia d'execució d'*sbatch* és la crida a una funció(*gen_script*). Aquesta funció crea l'*script* que es llançarà a cues, i retorna el nom d'aquest *script*.

4.5 Execució de les proves

Per analitzar el funcionament del *plugin* s'han utilitzat tres escenaris diferents, segons l'ample de banda de memòria consumit per la majoria d'aplicacions.

- HIGH: La majoria d'aplicacions (80%) tenen una gran demanda d'ample de banda de memòria.
- MEDIUM: Aproximadament la meitat (50%) de les aplicacions tenen uns requisits elevats d'ample de banda de memòria

```
sleep 18;
system("sbatch␣-n4␣--time=00:43␣--mem-bandwidth=88␣--lc-threshold=1.1␣".
    gen\_script("mg.C.4.3"));
sleep 28;
system("sbatch␣-n2␣--time=04:27␣--mem-bandwidth=1935␣--lc-threshold=1.1␣
    ". gen\_script("sintetichigh.16"));
sleep 187;
system("sbatch␣-n4␣--time=06:08␣--mem-bandwidth=1935␣--lc-threshold=1.1␣
    ". gen\_script("sintetichigh.22"));
sleep 38;
system("sbatch␣-n8␣--time=01:21:02␣--mem-bandwidth=1600␣--lc-threshold
    =1.1␣". gen\_script("cg.D.8.34"));
sleep 51;
system("sbatch␣-n1␣--time=01:23␣--mem-bandwidth=1935␣--lc-threshold=1.1␣
    ". gen\_script("sintetichigh.5"));
```

Figura 4.8: Fragment de l'*script submitter.pl*

- LOW: Poques aplicacions (10%) tenen un alt consum d'ample de banda de memòria

A partir d'aquests escenaris, s'han generat 3 workloads, un per a cada escenari, i s'han realitzat 6 execucions de cada un d'ells, variant el *plugin* de selecció de recursos o els paràmetres d'aquest. S'ha utilitzat el *plugin* original (Consumable Resources) i el *plugin* desenvolupat en aquest projecte, tenint sempre activat el paràmetre que permet augmentar el temps requerit per un treball. Les variacions doncs venen donades per l'activació o no del paràmetre que permet augmentar la penalització, i del valor del llindar màxim de penalització d'un treball. S'han descartat les combinacions amb la opció d'augmentar el temps sol·licitat per un treball per què aquest fet provocaria que es matessin molts treballs per excedir el temps sol·licitat. Les combinacions són les següents:

Threshold	**REQ_TIME**	**PENALTY**
Consumable Resources	-	-
Less Consume 1	X	-
Less Consume 1.1	X	-
Less Consume 1.1	X	X
Less Consume 1.2	X	-
Less Consume 1.2	X	X

Taula 4.1: Escenaris de les diverses proves realitzades

4.6 Avaluació dels resultats

A continuació s'analitzen els resultats de les execucions dels *workloads* en els diferents escenaris. En primer lloc es realitza un anàlisi entre les diverses execucions d'un mateix *workload* per comparar les diferències entre les diferents polítiques i, posteriorment, un anàlisi global del

comportament del *plugin*. Les mètriques utilitzades estan dissenyades per a validar els següents comportaments: L'augment del llindar màxim permès de penalització ha de comportar una reducció en el temps d'espera dels treballs, un augment en la penalització en el temps d'execució, un increment en la utilització i la saturació del sistema. Per fer referència a les diferents configuracions del workloads, en totes les imatges s'utilitzen abreviatures; en primer lloc el tipus de política de selecció de recursos, CR (Consumable Resources) o LC (Less Consume). A continuació, en els casos en que s'utilitzi un llindar, apareix el valor d'aquest (1.1 ò 1.2) i, finalment, apareix quin paràmetre de configuració s'ha utilitzat durant l'execució; LC_REQ_TIME (R) o LC_REQ_TIME_PENALTY(RP). També cal mencionar que, en les execucions s'ha configurat SLURM per a què permeti un cert temps extra d'execució de cada treball. Aquest temps, configurat a partir del paràmetre OverTimeLimit, és de 5 minuts, ja que és un valor acceptable i que permet que, en el cas de què a un treball li resti molt poc temps per finalitzar, tot el temps que s'ha executat no es perdi. La política de planificació que s'utilitza en les proves és la de *backfilling* que inclou SLURM.

4.6.1 Mètriques utilitzades

Per a la realització de l'anàlisi de les diferents execucions dels *workloads* en els diferents escenaris s'han creat una sèrie de gràfiques per poder visualitzar d'una manera més còmoda la gran quantitat d'informació de què es disposa i, d'aquesta manera centrar l'anàlisi cada vegada en un aspecte concret del funcionament. S'utilitzen les mètriques numèriques i anàlisis visuals següents:

Estat de les Aplicacions (gràfica paraver)

Aquest és l'anàlisi visual més general. La imatge extreta amb l'aplicació Paraver[Par], una eina desenvolupada al BSC-CNS, permet visualitzar per cada una de les execucions d'un *workload* en quin estat es troba cada aplicació. L'eix x mostra el temps i l'eix y les diferents aplicacions del *workload*. Així doncs, es disposa d'una línia per a cada aplicació que ens mostra la seva evolució. Aquestes mètriques estan normalitzades al temps d'execució més llarg, per tal de poder observar fàcilment la duració relativa entre les diferents execucions. S'observa el temps d'espera de cada treball, el temps d'execució i el temps total del workload. Una aplicació pot tenir 3 estats diferents, cada un representat amb un color diferent:

- Waiting: Aquest és l'estat en el què entra un treball al sistema. Es mostra en color vermell i pot no aparèixer si el treball comença a executar-se des del moment en el què s'envia al sistema de cues.

- Running: El treball es troba en aquest estat mentre s'està executant. Es mostra en color blau.

- Completed: Un cop ha finalitzat l'aplicació, pot aparèixer un espai en blanc que simbolitza el temps de més que ha sol·licitat el treball, és a dir, el temps que li ha sobrat respecte al demanat. Un cop passat aquest temps, la resta de la línia es mostra en negre.

Ús de l'ample de banda de cada node(gràfica paraver)

Aquesta imatge mostra l'ample de banda assignat per treballs a cada node. És la suma de l'ample de banda requerit per les tasques que s'estan executant en el node. L'eix x representa el temps i l'eix y són els nodes del sistema, tenint així una línia per a cada node. Els valors de l'ample de banda es mostren com un gradient de colors que van del verd al blau, de menys ample de banda de memòria a més respectivament. El color taronja indica que un node té més ample de banda assignat del que disposa, i per tant hi haurà penalització en els treballs que s'executen en aquest node. Qualsevol valor que superi els 3GiB/s es mostra en color taronja. El color negre indica un valor de 0 i, per tant, que no hi ha cap treball executant-se en aquest node.

Ample de banda del sistema

Aquesta mètrica és un gràfic de l'ample de banda de memòria assignat a tot el sistema. És la suma de l'ample de banda de memòria assignat a cada node en cada moment. L'eix x representa el temps i l'eix y la quantitat d'ample de banda de memòria assignat. Es presenten dues línies, una de color vermell que representa l'ample de banda en el cas de l'execució del *workload* amb CR i una línia de color blau que representa l'ample de banda de memòria de l'execució amb LC1.2RP. S'han escollit aquestes dues configuracions perquè són les que més saturació provoquen en els nodes. Aquesta mètrica permet obtenir una quantificació de la saturació del sistema que amb l'anterior imatge no es pot extreure, ja que el color taronja no ens indica quant ample de banda per sobre del límit del node està assignat.

Slowdown mig del temps d'execució

Aquesta mètrica permet comparar quin *slowdown*(penalització en el temps d'execució) obtenim amb cada una de les execucions d'un *workload*. A l'eix x es representen les diferents execucions, una barra per cada execució, mentre que a l'eix y hi ha el valor de l'*slowdown*. L'*slowdown* es calcula mitjançant la divisió del temps d'execució de cada treball respecte el temps requerit. Atès que els temps requerits estan molt ajustats per poder analitzar les diferències entre les diferents polítiques, l'*slowdown* ofereix una idea de quina execució ha patit més penalització. Es mostren dues barres per a cada execució. La blava indica l'*slowdown* mig de tot el *workload*, mentre que la línia taronja indica el valor màxim i mínim de l'*slowdown*. La fórmula utilitzada per calcular l'*slowdown* és:

$$Slowdown = \frac{TempsExecució}{TempsSol{\cdot}licitat}$$

Temps d'espera mig

Aquesta mètrica permet observar el temps mig d'espera dels treballs. A l'eix x es mostren les diferents configuracions i a l'eix y es mostra el temps. El temps d'espera, és el temps que passa des de que s'envia el treball al sistema de cues fins que s'executa.

Estat de finalització de les aplicacions

Aquesta mètrica permet analitzar quin és el comportament de cada política i veure quantitativament l'estat de finalització dels treballs de cada execució. A l'eix de les x es representen les diferents execucions, mentre que a l'eix de les y es representa el nombre d'aplicacions. Per a cada execució es disposen 4 barres; la blava indica el nombre de treballs matats (Killed) per excés de temps, la taronja, el nombre de treballs que han excedit el seu temps requerit, però que han finalitzat dins dels 5 minuts addicionals (OverTime). La barra groga indica el nombre de treballs que han finalitzat correctament (Completed) i, finalment la barra verda indica el nombre d'aplicacions que han fallat (Failed). Aquest últim cas només es dóna en les execucions de Less Consume, ja que els *workloads* contenen dos treballs de 32 cpus amb una alta demanda d'ample de banda de memòria. Aquest fet provoca que Less Consume no sigui capaç d'assignar un conjunt de cpus que satisfaci les necessitats d'ample de banda dels treballs mantenint la penalització màxima per sota del llindar estipulat pel treball. Això es podria sol·lucionar considerant el nombre de cpus sol·licitades pel treball com a un màxim i no com un requisit.

Percentatge d'ús de les CPUs

Aquesta mètrica ens permet analitzar quina és la utilització de les cpus del sistema durant l'execució del *workload*. A l'eix de les x es representen les diferents execucions amb una barra per cada una i, a l'eix de les y es representa el percentatge d'utilització de les cpus. Aquesta mètrica s'extreu calculant la mitjana de la mitjana d'ús de cada cpu.

Pencentatge d'útilització útil de les CPUs

Aquesta mètrica permet observar el percentatge d'ocupació del sistema per aquells treballs que finalitzen correctament. Es considera que un treball finalitza correctament si acaba la seva execució dins del temps requerit.

Ús de cada CPU (gràfica paraver)

Aquesta és també una mètrica extreta amb Paraver. De la mateixa manera que les anteriors, a l'eix de les x es representa el temps, mentre que a l'eix de les y es representen les cpus, una línia per cada una. Quan una cpu està assignada a un treball es representa en blau, mentre que si no està assignada es representa en negre. Amb aquesta mètrica es pot observar com es reparteixen les cpus i com a mesura que s'incrementa el llindar de penalització l'ús de les cpus augmenta.

4.6.2 Escenari HIGH

En aquest escenari és en el que es pot apreciar més diferència entre les diferents polítiques de selecció de recursos. Com es pot veure a la figura 4.9, hi ha una gran diferència entre el temps d'execució del workload CR respecte dels altres. Com és d'esperar, la primera execució del workload (CR) és la que menys temps d'execució té, mentre que l'execució més llarga és la LC. També es pot apreciar com es va reduint el temps d'execució total a mesura que s'augmenta el llindar màxim de penalització. També s'observa que a l'execució de CR gairebé no hi ha temps sobrant de les aplicacions (color blanc). Això és degut a què, en la majoria dels casos les aplicacions consumeixen tot el seu temps, o excedeixen el temps sol·licitat. Si una aplicació excedeix en 5 minuts el temps sol·licitat, temps addicional proporcinat amb el paràmetre de configuració OverTimeLimit, SLURM la mata(kill).

En el cas de les execucións de Less Consume també es pot apreciar un augment considerable del temps d'espera respecte a l'execució de CR. Això és degut a què, en aquest escenari, en haver-hi moltes aplicacions amb una gran demanda d'ample de banda, la política Less Consume fa esperar una aplicació fins que la compartició d'ample de banda de memòria no suposi una penalització superior al llindar màxim permès. Per tant, com es pot apreciar a les traces presentades, el temps d'espera es va reduïnt a mesura que s'augmenta el llindar de penalització.

A la figura 4.10, es pot observar l'ample de banda assignat a cada un dels nodes. Es pot apreciar clarament que l'execució CR no té cap mena de control sobre l'ample de banda de memòria assignat, i gairebé en tot moment els nodes es troben sobrecarregats. Aquest fet contrasta amb l'execució LC, en la que en cap moment hi ha cap node sobrecarregat, com era d'esperar doncs LC sí que realitza un control sobre l'ample de banda de memòria de cada node i treball.

A mesura que s'augmenta el llindar permès es pot apreciar com es van saturant els nodes fins al cas de l'execució LC1.2RP en la què pràcticament s'arriba a uns nivells de saturació semblants als de l'execució CR.

Tot i que en aquesta gràfica no es pot apreciar, a la figura 4.11 es pot veure un resum de l'ample de banda assignat a tot el sistema. D'aquesta manera es pot observar com l'ample de banda assignat de l'execució CR és molt superior al de l'execució LC1.2RP. Això provoca una

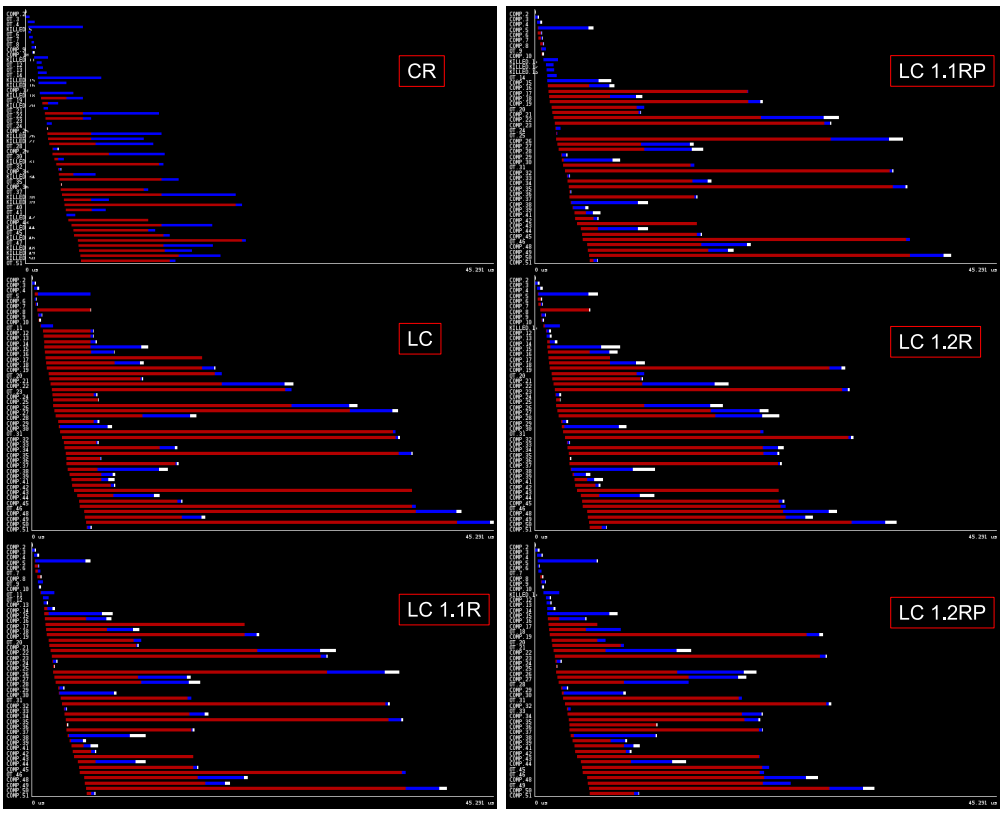

Figura 4.9: HIGH: Vista del procés de les aplicacions en les diferents execucions del *workload*.

penalització en els treballs molt més elevada en el cas de CR que en el cas de LC1.2RP. També es pot observar que l'ample de banda de memòria de tot el sistema en el cas de LC1.2RP no arriba mai a 30000MB/s mentre que l'execució de CR supera aquesta xifra durant gran part de la seva execució.

A la figura 4.12 es pot veure un resum de l'*slowdown* de l'execució de les aplicacions respecte al temps d'execució sol·licitat i el temps mig d'espera dels treballs. En aquest cas es pot observar com la mitjana d'*slowdown*(barra blava) és molt superior en el cas de CR respecte a les execucions de Less Consume. També el màxim i mínim(barra taronja) són molt superiors en el cas de l'execució de CR que en els altres casos, aproximadament el doble. Un fet curiós és que l'*slowdown* màxim obtingut per les execucions de Less Consume supera el llindar establert en cada una de les execucions. Aquest fet és degut a que l'ample de banda de memòria no és l'únic factor determinant a l'hora d'executar una aplicació, ja que la distribució de les tasques en els diferents nodes o l'ample de banda de xarxa també són factors que s'haurien de tenir en compte i que poden provocar que les aplicacions s'executin més lentament de l'esperat. La utilització del paràmetre que permet augmentar la penalització, mostra un augment de l'slowdown mig dels treballs.

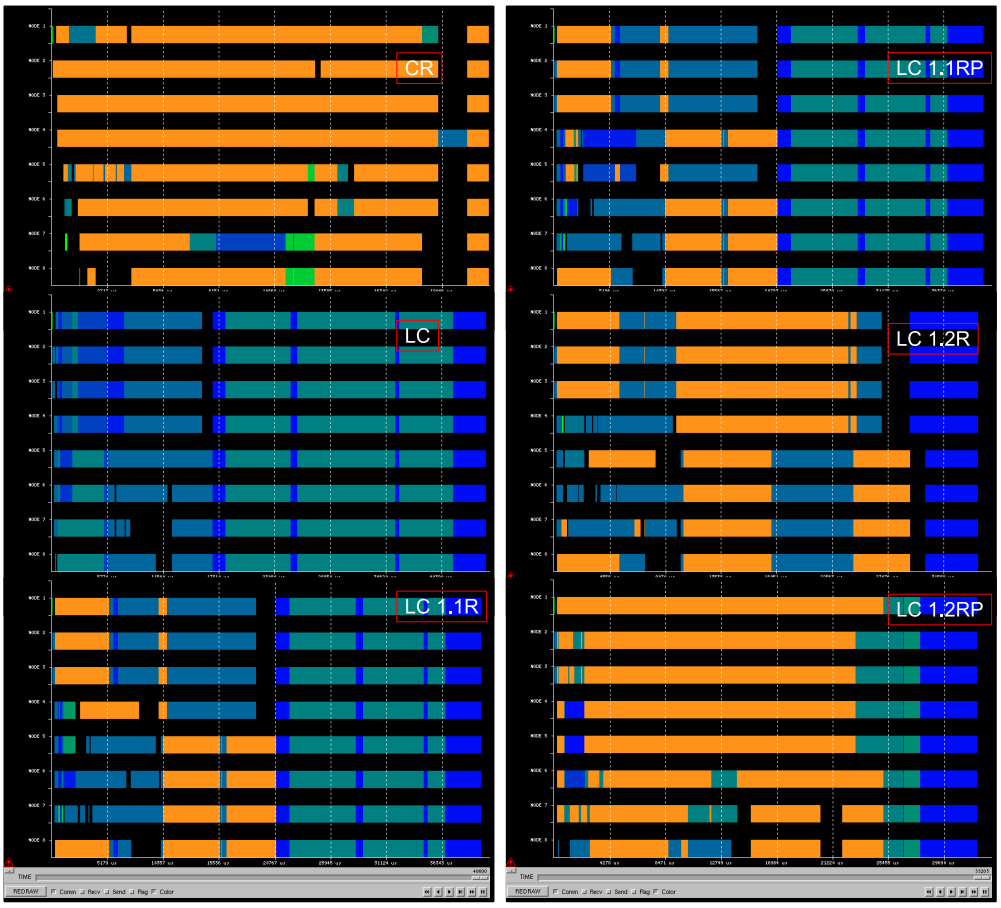

Figura 4.10: HIGH: Vista de l'ús d'ample de banda de memòria de cada Node en les diferents execucions.

També es pot observar com l'ús de Less Consume dispara el temps d'espera mig dels treballs i com la utilització dels llindars de penalització redueixen considerablement aquest temps. Teambé es pot observar una disminució del temps d'espera utilitzant el paràmetre que permet augmentar la penalització dels treballs ja començats.

Una altra dada a tenir en compte a l'hora de comparar les diferent execucions és l'estat de finalització dels treballs. A la figura 4.13 es pot veure un resum de l'estat en el què han acabat els diferents treballs executats en les diferents execucions del *workload*. Es pot observar que l'execució de CR té un gran nombre d'aplicacions que han hagut de ser aturades per SLURM per haver excedit els 5 minuts extres respecte el temps sol·licitat(franja blava). En la resta d'execucions gairebé no hi ha cap aplicació que s'hagi hagut de matar.

També s'observa un gran nombre de treballs que han finalitzat dins dels 5 minuts extres(franja taronja), mentre que en les execucions de Less Consume gairebé no n'hi ha, excepte en el cas de LC1.2RP. Cal remarcar l'aspecte negatiu que apareix en totes les execucions de Less Consume, això és, que aquestes mostren dos treballs que han fallat(franja verda). Aquests

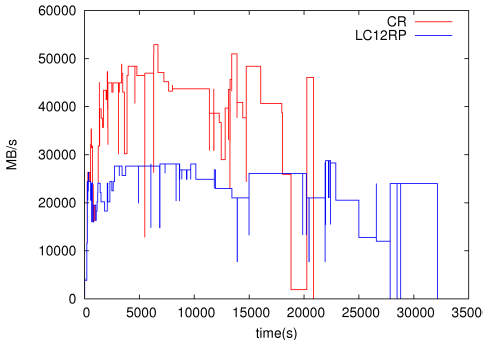

Figura 4.11: HIGH: Vista de l'ús d'ample de banda de memòria del sistema en dues execucions.

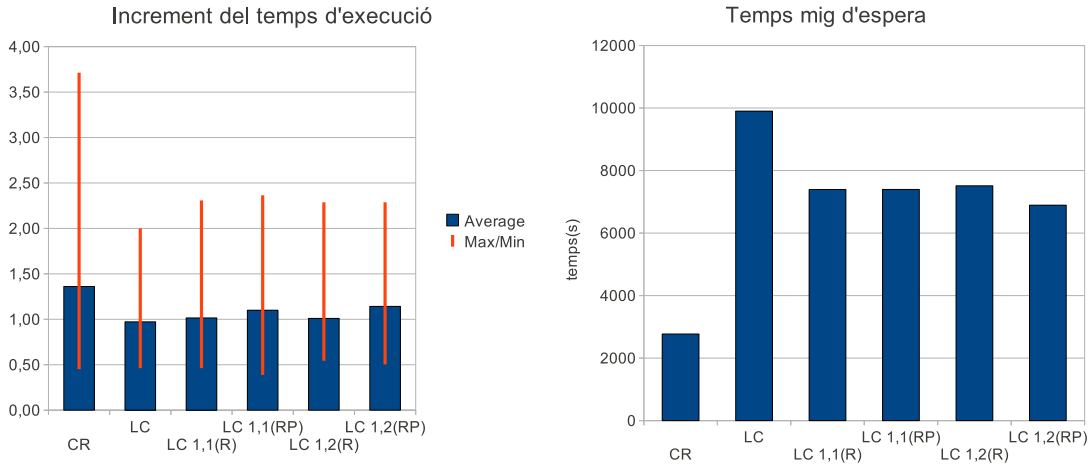

Figura 4.12: HIGH: Slowdown en cada una de les execucions

són dos treballs que sol·liciten 32 cpus i un ample de banda de memòria elevat, el què provoca que Less Consume no sigui capaç de trobar una assignació vàlida per a aquests dos treballs que no superi el llindar màxim permès de penalització, i per tant rebutja els dos treballs.

Un fet no esperat és l'aparició de treballs matats en les execucions de Less Consume. Tot i que en les dues primeres execucions (LC i LC1,1R) no s'aprecia cap treball matat, a les tres execucions següents sí que apareixen aplicacions que s'ha matat per excedir-se del temps requerit. Aquest fet ve donat per la compartició d'altres recursos apart de l'ample de banda de memòria, com pot ser l'ample de banda de xarxa, i també per la distribució de les tasques en els nodes, ja que una aplicació no es comporta de la mateixa manera si es col·loquen dues tasques per node o només una.

Es pot observar també que en les configuracions en què es permet augmentar la penalització dels treballs ja començats, augmenten el nombre de treballs que excedeixen el seu temps requerit respecte a les configuracions que no permeten augmentar la penalització.

A la figura 4.14, es pot observar el percentatge mig d'ús de les cpus durant tota l'execució i

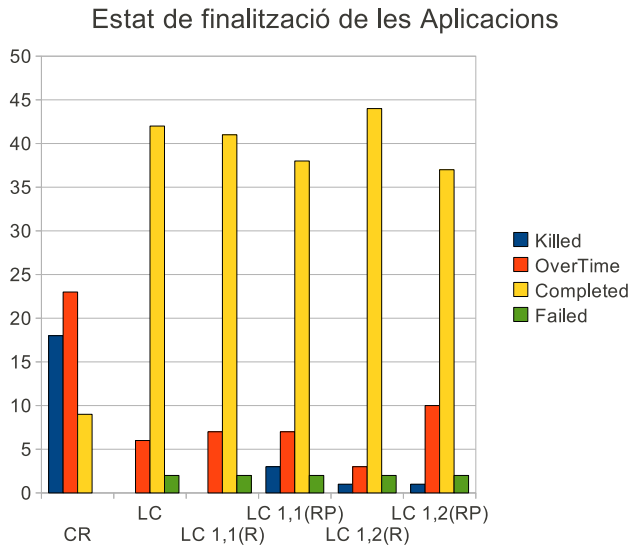

Figura 4.13: HIGH: Estat de finalització dels treballs en cada una de les execucions del workload

el percentatge d'utilització per part dels treballs que finalitzen dins del temps sol·licitat. Com era d'esperar, l'execució de CR és la que mostra un percentatge més alt d'utilització de les cpus, mentre que l'execució de LC és la que menys en té. Es pot apreciar, de la mateixa manera que en altres mètriques, com a mesura que s'augmenta el llindar de penalització, el percentatge d'utilització augmenta considerablement, des del 30% de LC fins al 45% de LC1.2RP. Així i tot, però, el que no era d'esperar és que les les dues execucions amb un llindar de 1.1 (LC1.1R i LC1.1RP) tinguin el mateix valor. Això és degut als grans requeriments d'ample de banda de memòria dels treballs que provoquen que, encara que es permeti augmentar la penalització d'un treball un cop iniciat, no es pugui assignar un processador d'un node ja assignat, ja que això provocaria superar el llindar establert.

A la figura 4.15, es pot veure la distribució de les cpus assignades a cada moment. Es pot destacar l'alta assignació de l'execució CR i la baixa de l'execució LC, on pràcticament durant tota l'execució del *workload*, cada node tan sols té una cpu assignada. Entre les execucions LC1.1R i LC1.1RP no s'observen pràcticament canvis significatius com ja s'ha explicat anteriorment, mentre que entre les execucions LC1.2R i LC1.2RP sí que s'aprecien diferències, encara que aquestes són pràcticament mínimes, i la darrera té menys temps amb nodes amb una sola cpu assignada.

4.6.3 Escenari MEDIUM

Aquest escenari està format per, aproximadament, la meitat dels treballs de grans necessitats d'ample de banda de memòria i l'altra de meitat de poques necessitats. Aquest fet es pot

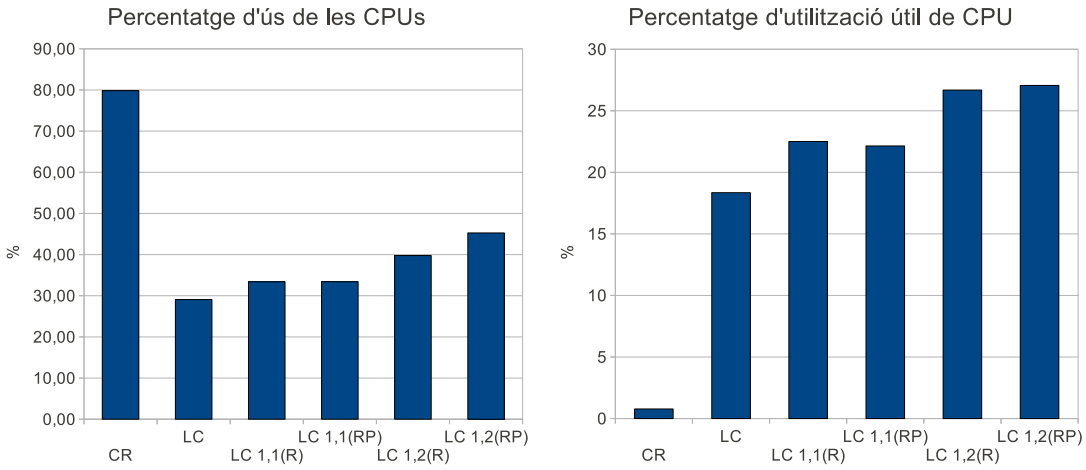

Figura 4.14: HIGH: Percentatge d'utilització de les CPUS

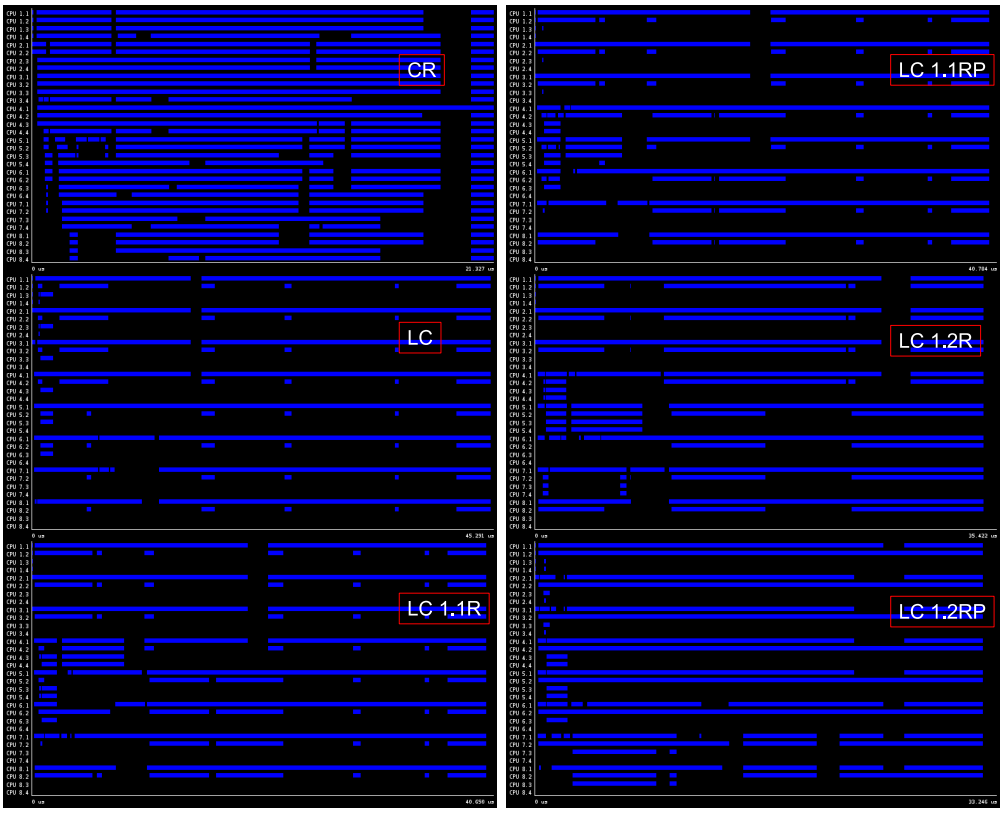

Figura 4.15: HIGH: Vista de l'ús de cada CPU en les diferents execucions del workload.

constatar amb la figura 4.16, ja que en ella es pot veure que no hi ha tanta diferència entre les execucions. La que més triga és la LC amb un temps total de 24.858 segons, que és practicament el mateix temps de les execucions LC1.1R i LC1.2R. Cal destacar igualment, el gran nombre

d'aplicacions que no arriben a consumir tot temps sol·licitat, sobretot en les execucions LC1.2. Aquest fet és degut a què l'augment realitzat sobre el temps requerit és superior a la penalització final dels treballs, el què provoca que els acabi sobrant temps. També cal destacar un curios augment del temps d'espera de tres aplicacions: 16,17 i 19. No s'ha pogut esbrinar el perquè d'aquest fet, però el més segur és que sigui degut al mecanisme de backfilling d'SLURM.

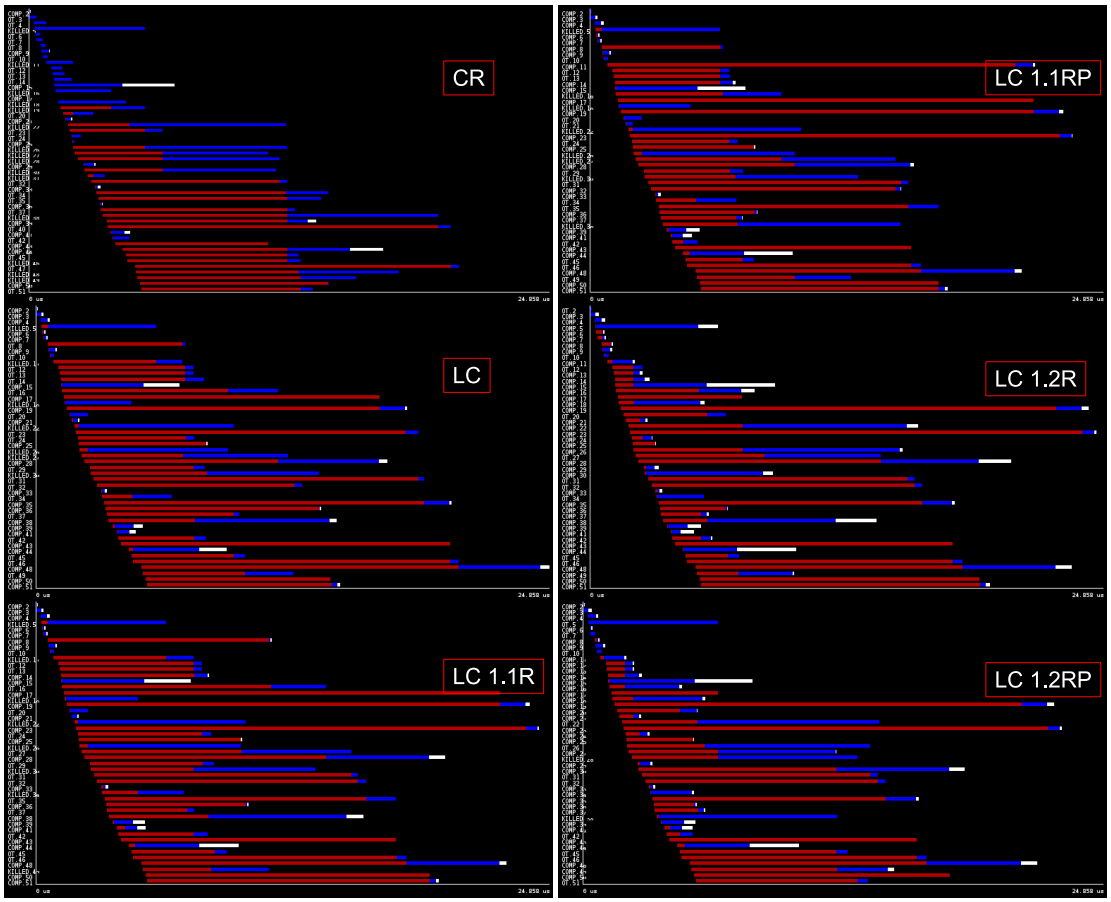

Figura 4.16: MEDIUM: Vista del proces de les aplicacions en els diferents workloads

A la figura 4.17, es pot apreciar l'assignació de l'ample de banda de memòria en els diferents nodes. Cal destacar d'aquesta figura la gran saturació que s'observa en l'execució LC1.2RP que supera al temps de saturació de l'execució CR, tot i que aquesta no té en compte l'ample de banda de memòria. Així i tot, es pot observar com en la resta d'execucions, la saturació és relativament baixa, havent-hi fins i tot zones clarament verdes(baix consum d'ample de banda de memòria).

La figura 4.18 mostra l'ample de banda de memòria acumulat del sistema en les dues execucions CR i LC1.2RP. Com es pot observar, l'execució de CR està gairebé sempre per sobre de la de LC1.2RP, que no supera mai els 30000MB/S. Així i tot, l'execució de LC1.2RP no dura

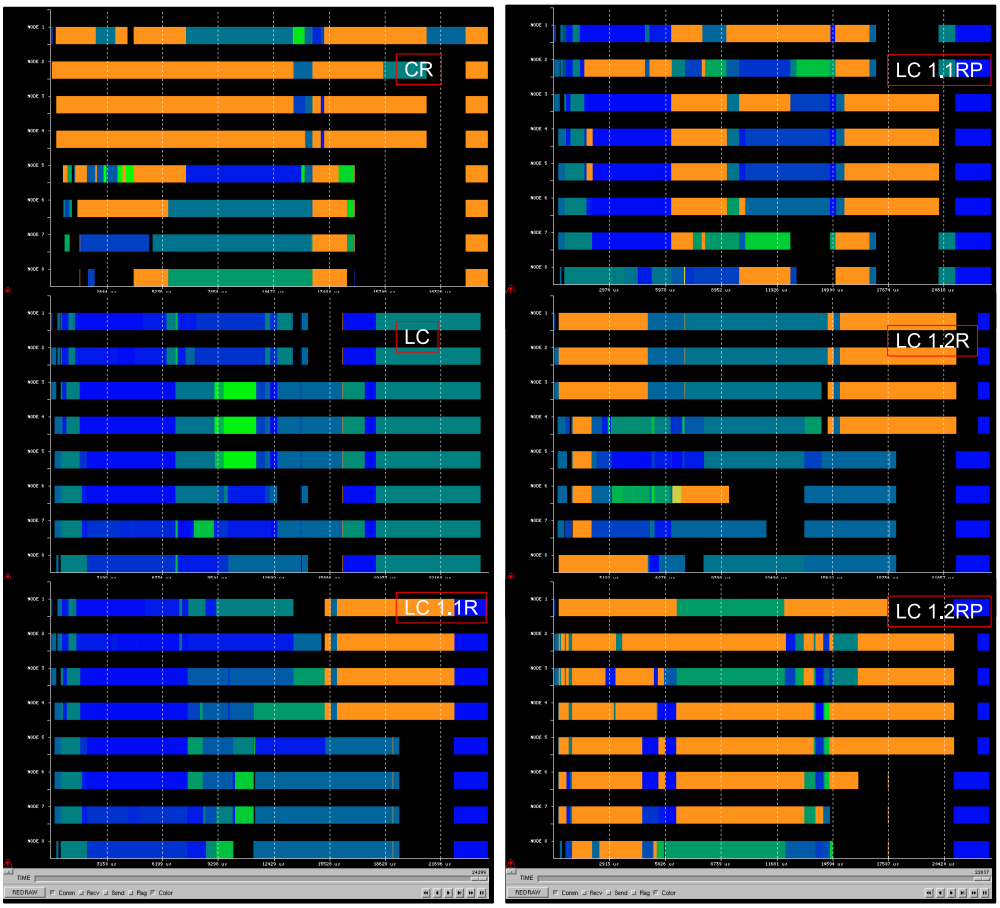

Figura 4.17: MEDIUM: Vista de l'ús d'ample de banda de cada Node en els diferents workloads.

gaire més que la de CR (aproximadament 3000 segons més).

A la figura 4.19 podem observar com l'*slowdown* de les execucions és bastant semblant, tot i que l'execució de CR és clarament superior a la resta, de la mateixa manera que el seu màxim. També es pot observar, a la gràfica de temps mig d'espera, com l'augment del llindar i l'activació del paràmetre que permet augmentar la penalització tenen un impacte important en el temps d'espera mig dels treballs. Un fet curios, és el cas de l'execució de LC, que té un slowdown superior a 1 quan en aquesta execució cap treball té penalització. Aquest fet és degut al gran ajustament dels temps requerits pels treballs per tal de poder extreure informació que permeti realitzar una comparació més clarament entre les diferents execucions.

Pel que fa a la finalització dels treballs, a la figura 4.20 es pot observar com l'execució de CR té un gran nombre d'aplicacions matades per excedir el temps requerit. També s'observa un gran nombre d'aplicacions que han excedit el temps requerit sense haver hagut de ser matades. Com s'ha comentat anteriorment el nombre d'aplicacions matades ve donat pel gran ajustament en els temps requerits pels treballs. Aquest fet és el que justifica l'aparició de treballs matats

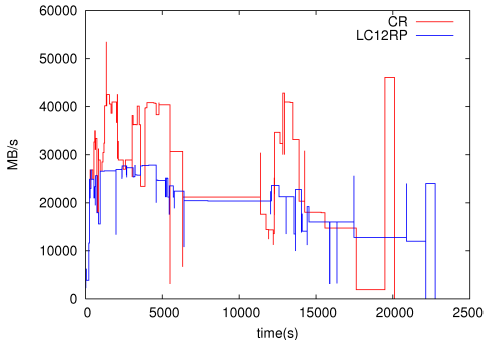

Figura 4.18: MEDIUM: Vista de l'ús d'ample de banda de memòria del sistema en dues execucions.

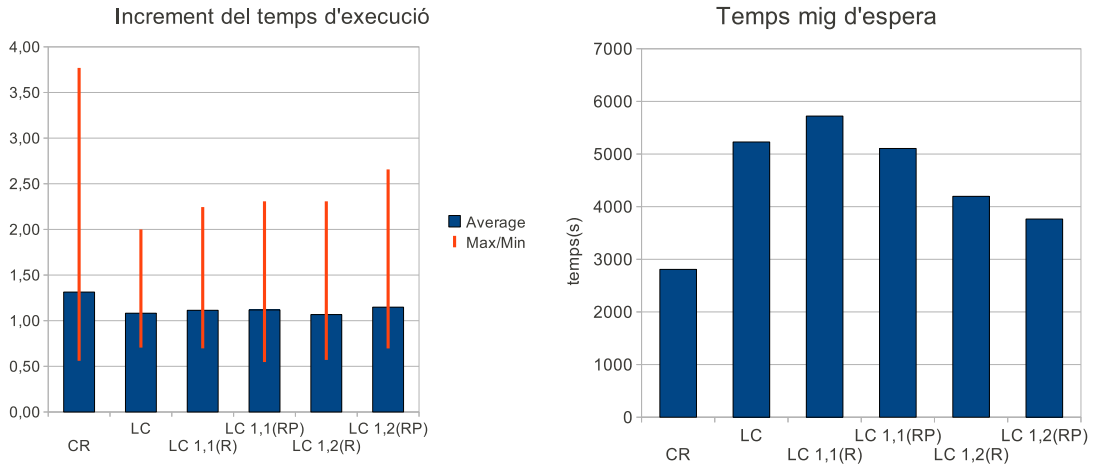

Figura 4.19: MEDIUM: Slowdown en cada una de les execucions

en l'execució de LC. Un fet a destacar és que a l'execució de LC1.2R no hi ha cap aplicació que s'hagi hagut de matar. Això és degut a què l'augment del temps requerit és superior a la penalització dels treballs el què provoca que realment aquests disposin s'un temps addicional per executar-se. Es pot observar també, com el fet d'augmentar el temps sol·licitat provoca que hi hagi més treballs que finalitzen dins del seu temps i, com l'activació del paràmetre que permet augmentar la penalització provoca una dismunicó d'aquests treballs en el cas LC1.2RP.

El fet més a destacar d'aquest escenari és la relativa igualtat en la utilització de les cpus. Com es pot veure a la figura 4.21, l'ús de les cpus és molt igualat entre les diferents execucions tot i que l'execució de CR té una clara avantatge sobre les demés. Un fet curiós és que l'execució de LC és usperior a tres de les execucions de Less Consume. Aquest fet es produeix principalment pel temps d'espera de recursos que pateixen LC1.1R, LC1.1RP i LC1.2R per a poder executar un treball de 16 cpus. Aquest fet però no es pot controlar i podria haver estat completament a a inversa, però és un fet a destacar ja que va en contra del resultat esperat. Es pot observar també l'efecte que tenen les diferents configuracions en l'ús útil de cpu. Es pot observar l'augment

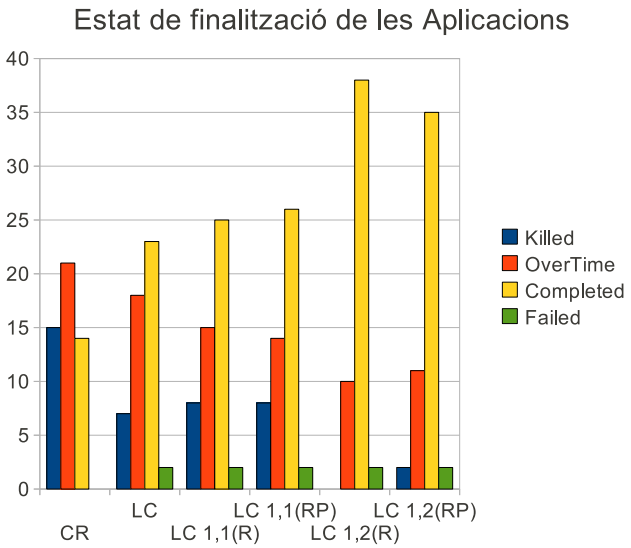

Figura 4.20: MEDIUM: Estat de finalització dels treballs

de la utilització a mesura que s'incrementa el llindar, però hi ha una baixada important quan s'activa el paràmetre que permet augmentar la penalització.

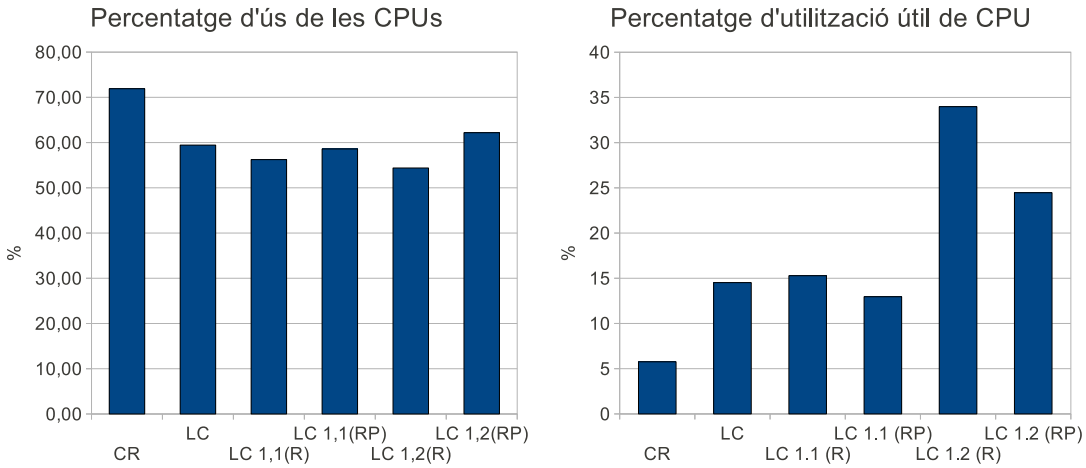

Figura 4.21: MEDIUM: Percentatge d'utilització de les CPUS

A la figura 4.22 es poden observar els fets abans esmentats. S'observa clarament el temps d'espera de recursos, sobretot a les execucions CR i LC1.2RP, tot i que les altres també en tenen, però en aquestes és més prounciat. També es pot observar com les execucions LC, LC1.1R i LC1.1RP són molt similars, i com era d'esperar el gran ús de cpus que s'aconsegueix en el cas de l'execució de CR. També s'observa que la primera meitat de totes les execucions de Less

Consume la utilització de les cpus és molt alta. Aquest fet és degut a que el backfilling és capaç de col·locar els treballs petits en els forats que queden lliures, i a mesura que els treballs petits es van acabant i només queden els més grans, a utilització de les cpus va disminuint.

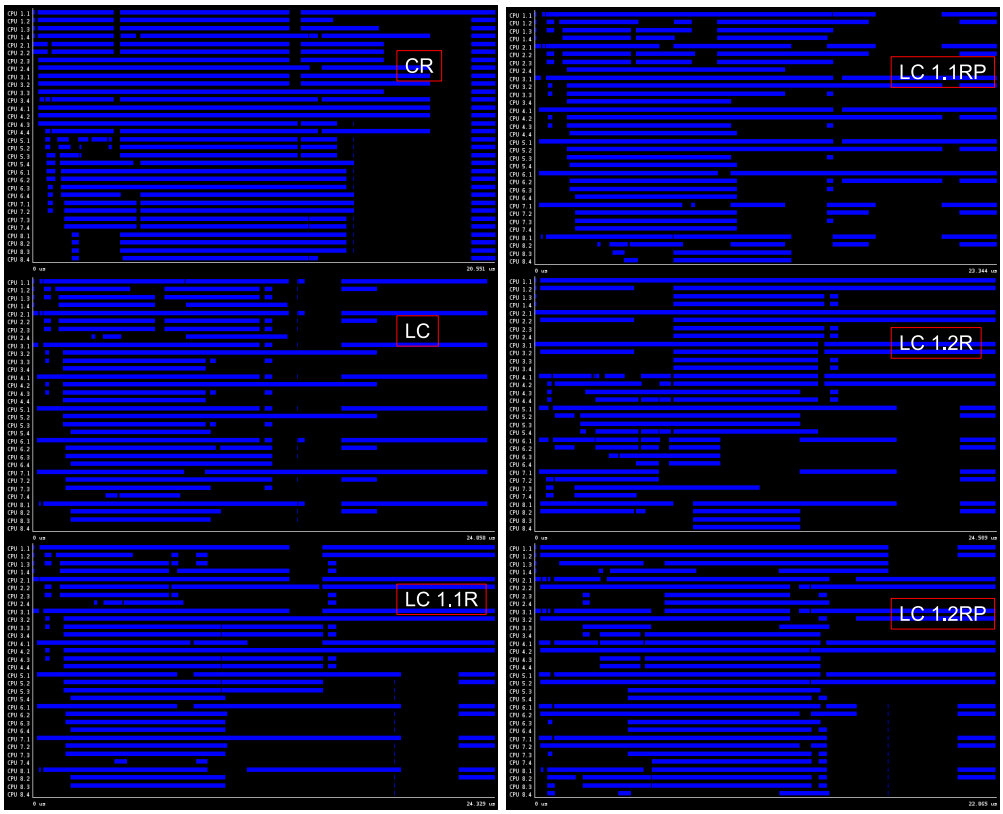

Figura 4.22: MEDIUM: Vista de l'ús de cada CPU en els diferents workloads.

4.6.4 Escenari LOW

En aquest escenari la majoria d'aplicacions tenen uns requisits molt baixos d'ample de banda de memòria. Aquest fet provoca que gairebé cap treball tingui penalització.

A la figura 4.23 es pot observar com totes les execucions del workload tenen una duració similar. Pot resultar extrany observar que l'execució que més triga en acabar és la de LC1.2R. Aquest fet és degut a que el treball 50 es passa molta estona esperant per a ser executat a causa de no tenir prous processadors lliures que estant essent utilitzats pels treballs 48 i 49. També es pot observar, de la mateixa manera que a l'escenari anterior, que hi ha molts treballs que acaben la seva execució abans del temps requerit, sobretot a les execucions LC1.2. Això ve donat perquè l'augment del temps requerit és superior a la penalització obtinguda pel treball, el que fa que aquest pugui executar-se a una velocitat normal, i per tant li sobri gairebé tot el temps addicional.

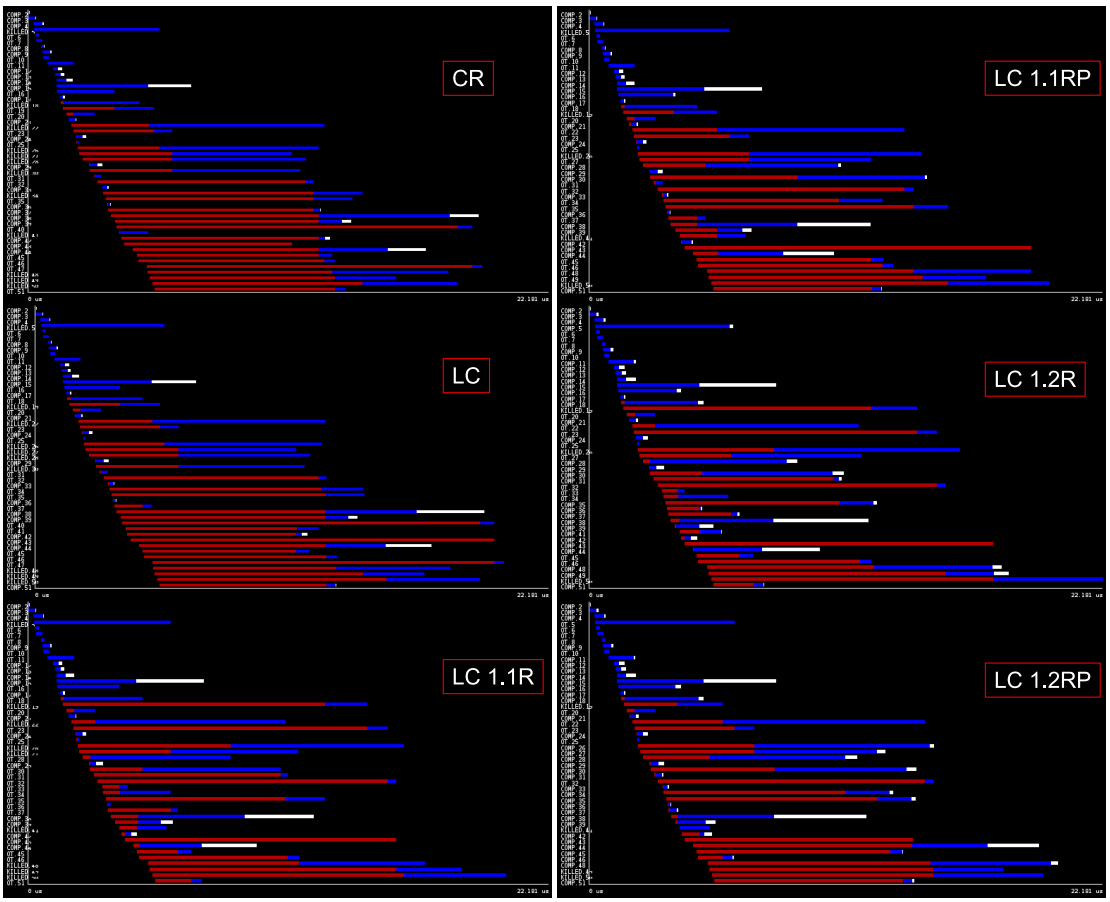

Figura 4.23: LOW:Vista del process de les aplicacions en els diferents workloads

Un altre tret a destacar d'aquestes execucions és que el temps d'espera és pràcticament el mateix a les execucions CR i LC, contràriament al que es podria esperar. Això està provocat per la gran cuantitat de treballs amb una demanda molt baixa d'ample de banda de memòria, fet que provoca que aquest paràmetre dels treballs pràcticament no tingui importància a l'hora de realitzar la selecció de recursos, ja que difícilment hi haurà penalització entre treballs.

Com es pot observar a la figura 4.24, l'ample de banda acumulat a cada node és molt baix, i fins i tot en el cas de l'execució de CR gairebé no hi ha saturació dels nodes. Com era d'esperar en aquest escenari, gairebé no hi ha saturació en cap cas, excepte a l'execució de LC1.2RP tot i que en aquesta la saturació té una durada força curta.

Com es pot veure a la figura 4.25, l'ample de banda acumulat del sistema és pràcticament el mateix en les execucions CR i LC1.2RP. De fet a la primera part de l'execució les dues gràfiques es solapen degut a que l'assignació d'ample de banda de memòria és exactament la mateixa en les dues execucions. Cal destacar, que tot i que a la gràfica anterior s'ha observat una saturació d'algun dels nodes del sistema, aquest fet no és observable amb aquesta mètrica doncs en cap

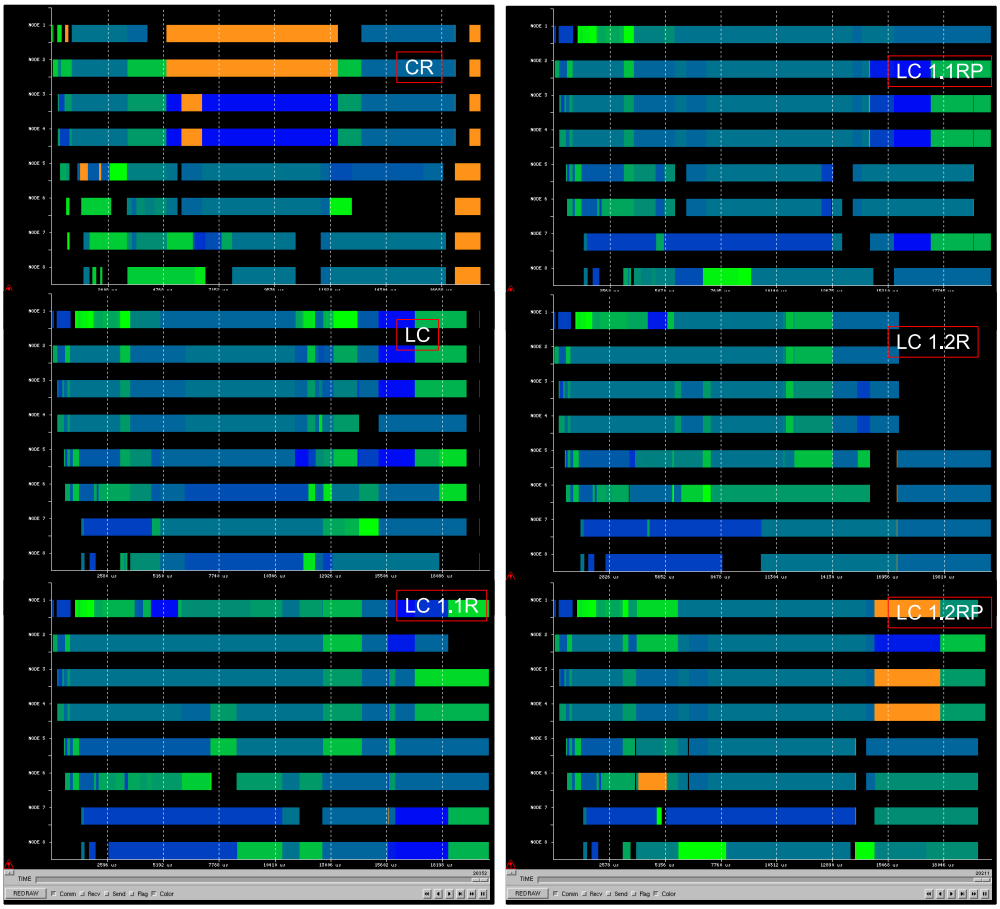

Figura 4.24: LOW:Vista de l'ús d'ample de banda de cada Node en els diferents workloads.

moment es sobrepassa l'acumulat d'ample de banda de memòria dels nodes.

Com és d'esperar en aquest escenari, l'slowdown de les execucions és pràcticament idèntic en tots els casos com mostra la figura 4.26, obtenint l'execució de CR fins i tot millors resultats que algunes de les execucions de Less Consume. En aquest escenari les diferències entre les diferents configuracions i valors del llindar màxim són pràcticament inexistents. Tot i així, hi ha una clara diferència entre el temps d'espera mig de l'execució amb LC i la resta que utilitzen Less Consume.

Pel què fa a la finalització dels treballs, però, els resultats no són exactament els esperats. Com es pot observar a la figura 4.27 hi ha un gran nombre de treballs matats, sobretot en l'execució de CR i, també hi ha un gran nombre d'aplicacions que s'han excedit del temps sol·licitat. Aquest fet demostra que l'ample de banda de memòria no és l'únic element que pot provocar un ralentiment de l'execució dels treballs. De la mateixa manera que a l'escenari anterior, l'augment del temps sol·licitat en el cas de les execucions LC1.2 permet que acabin moltes més aplicacions que no pas en les altres execucions.

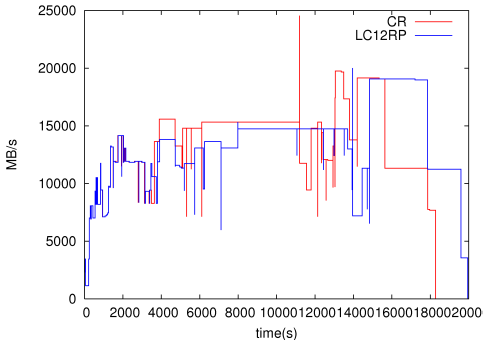

Figura 4.25: LOW: Vista de l'ús d'ample de banda de memòria del sistema en dues execucions.

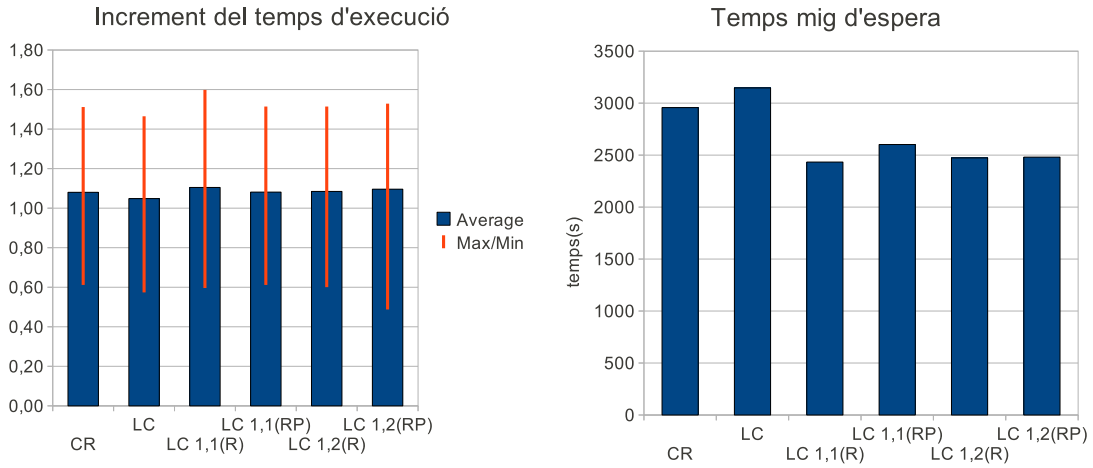

Figura 4.26: LOW: Slowdown en cada una de les execucions

Com també era d'esperar, la utilització de les cpus és força alta com es pot observar a la figura 4.28, arribant al 86% en el cas de l'execució de CR. El cas a destacar és el de l'execució de LC1.2R que està molt per sota de la resta amb un 76% d'ús de cpus. Aquest fet s'explica, com es pot veure a la figura 4.29, perquè cap al final la meitat dels processadors estan parats esperant a que acabi el treball que està situat a la segona meitat del sistema. Aquest fet és el que provoca que la mitjana d'ús de les cpus baixi considerablement en aquesta execució del workload. Es pot observar clarament en la gràfica d'utilització útil de cpu, com l'augment del llindar tenen un clar impacte en l'ús util de les cpus. Quant més s'augmenta el llindar més gran és la utlització útil del sistema, donat que la penalització és molt inferior al llindar però sempre s'aplica l'augment del temps sol·licitat.

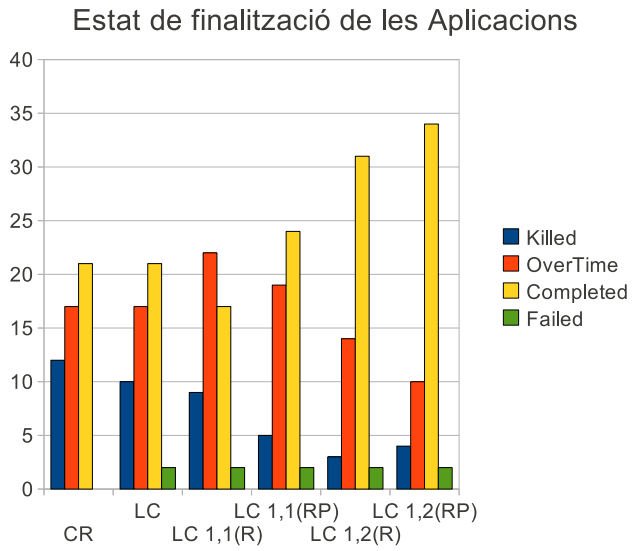

Figura 4.27: LOW: Estat de finalització dels treballs

4.6.5 Anàlisi Conjunt

Com s'ha vist a les seccions anteriors hi ha molta diferència en el comportament de les diferents polítiques de selecció de recursos. En tots els escenaris la política CR és la més ràpida en executar tots els treballs, però també és amb la que es maten més treballs i amb la que acaben més treballs dins del temps addicional que proporciona SLURM als treballs. Si no fos per aquest temps en el cas de l'escenri HIGH pràcticament tots els treballs es matarien per excedir el temps sol·licitat. Aques però no és el cas de les execucions amb Less Consume. Si bé aquestes són molt més lentes que Consumable Resources, la quantitat de treballs que es maten o que acaben fora del seu temps són moltes menys.

Un fet que cal tenir en compte i que juga a favor de la política Consumable Resources és la utilització de les cpus. Tal i com es realitza actualment la comptabilitat d'hores en els centres de supercomputació, el més important és omplir al màxim el sistema, ja que es factura per hora de cpu utilitzada però no per hora de cpu útil. Per a poder aplicar Less Consume en un entorn real caldria aplicar nous models de comptabilitat d'hores basats en la utilització de recusos i no únicament en hores de cpus. És a dir, si un treball sol·licita tot l'ample de banda de memòria d'un node per cada una de les seves tasques, aquest treball realment està utilitzant tot el node, doncs està evitant que altres treballs puguin executar-se en aquell node i, per tant s'hauria de comptabilitzar que realment utilitza tot el node i no tansols una cpu.

Com s'ha pogut comprovar en les anterior seccions, la penalització en el temps d'execució es manté practicament invariant en els diferents escenaris utilitzant Less Consume mentre que la utilització de Consumable Resources obté una penalització mitja superior a les altres execucions.

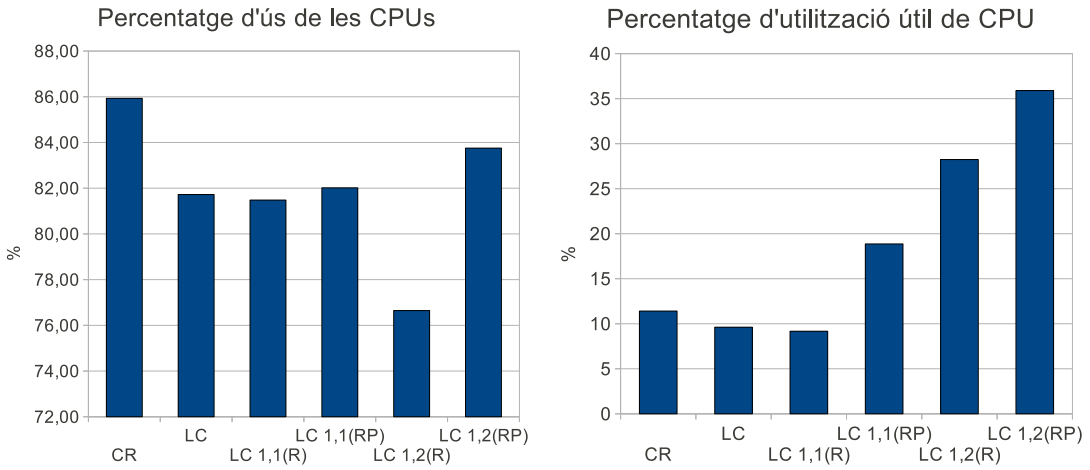

Figura 4.28: LOW: Percentatge d'utilització de les CPUS

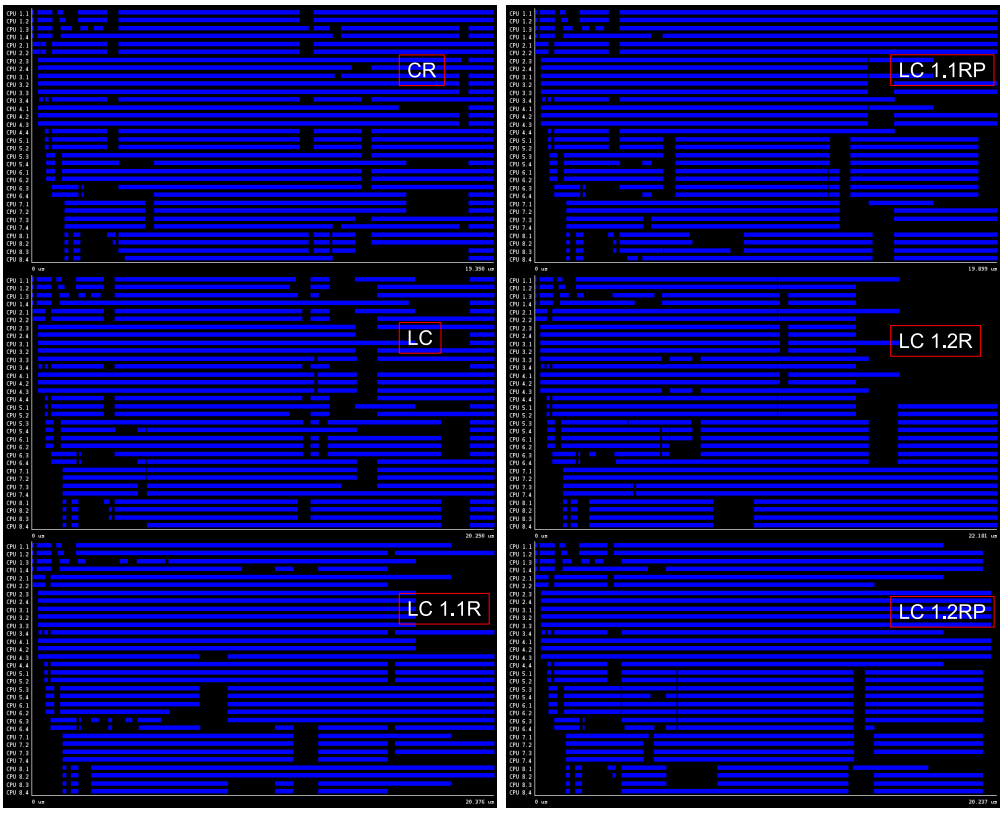

Figura 4.29: LOW: Vista de l'ús de cada CPU en els diferents workloads.

També es pot observar com els temps d'espera de Consumable Resources són inferiors en els casos en que hi ha saturacions en els nodes i com l'augment del llindar màxim de penalització i el pemetre l'augment de la penalització dels treballs ja executant-se provoquen una disminució

del temps d'espera en tots els escenaris.

Tot i que en la utilització de cpu, Consumable Resources és molt superior a Less Consume, en la utilització útil de cpu Less Consume es comporta millor. En tots els casos la majoria d'execucions de Less Consume són clarament superiors en quant a percentatge d'utilització útil de la cpu, essent les execucions amb un llindar més elevat les que obtenen els millors resultats.

Així doncs podem concloure, que la millor combinanció de llindar màxim de penalització i de paràmetres de configuració de la política és Less Consume amb un llindar de 1,2 i el paràmetre de configuració LC_REQ_TIME_PENALTY, ja que ofereix una disminució important del temps d'espera respecte a Less Consume sense llindar, la penalització en el temps d'execució és pràcticament la mateixa que Less Consume i el percentatge d'utilització útil de les cpus és clarament superior.

CAPÍTOL 5

Planificació i Anàlisi econòmic

L'objectiu d'aquest capítol és donar una idea del cost del treball realitzat durant el projecte. La primera part mostra la planificació del projecte per a poder comptabilitzar les hores dedicades i, per altra banda el cost dels equipaments utilitzats per obtenir finalment el cost econòmic del projecte.

5.1 Planificació

A continuació es fa un repàs de la planificació temporal del projecte. A la figura 5.1 es pot veure aquesta planificació. Té una resolució setmanal, i va des de finals de setembre del 2009 fins a mitjans de juny del 2010. S'han dedicat 4 hores diàries, excepte a la última fase del projecte on la dedicació ha estat superior. Al diagrama es pot apreciar visualment l'evolució de les diferents fases del projecte. S'ha dividit el projecte en 4 gran etapes: la concepció del projecte, el desenvolupament, l'anàlisi i la documentació del projecte.

La primera fase consisteix principalment en la recerca de informació, però també inclou un estudi de la viabilitat d'implementació del projecte dins d'SLURM i la manera de fer-ho.

La fase de desenvolupament consisteix, en l'implementació dels requeriment previs per poder implementar la política Less Consume, la implementació de la política bàsica i el plugin d'accounting i finalment la implementació de les millores. Totes les tasques d'aquesta fase inclouen també les proves respectives per a garantir un correcte funcionament.

La fase d'avaluació inclou la preparació de l'entorn per poder realitzar les proves, l'execució i tria de les aplicacions a executar dins els workloads i l'execució i anàlisi dels workloads en els

diferents escenaris.

Finalment, la fase de documentació, inclou la redacció de l'informe previ, lliurat el dia 10 d'abril i la redacció d'aquesta memòria.

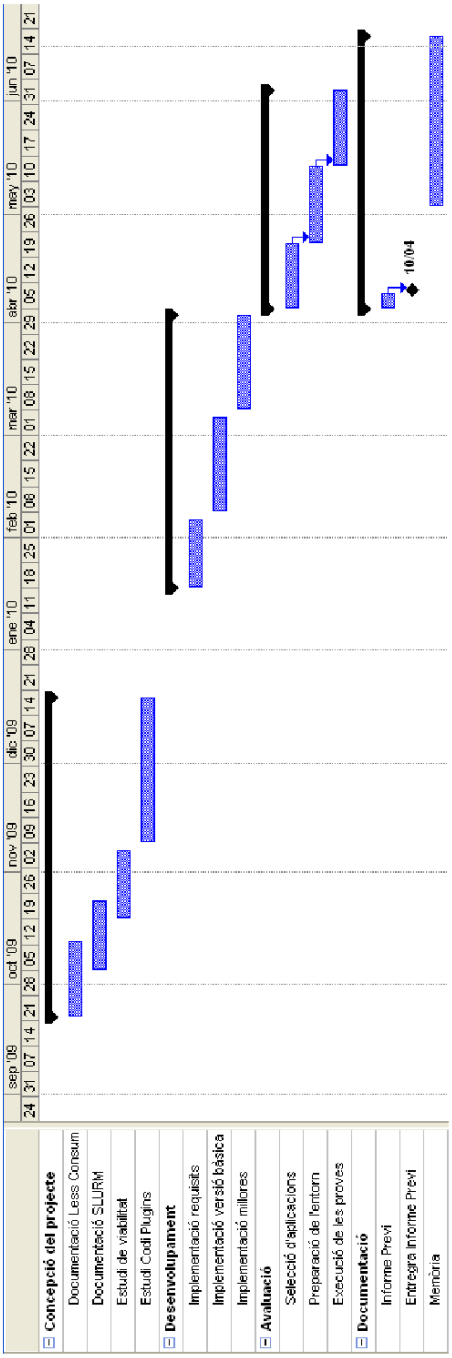

Figura 5.1: Planificació temporal del projecte

5.2 Anàlisi econòmic

El cost del projecte està dividit en dues parts. Per una banda hi ha el cost de la implementació dels dos plugins desenvolupats i de la preparació de l'entorn i, per l'altra banda hi ha el cost de l'equipament on s'han realitzat les proves.

Cost d'implementació

Per a comptabilitzar el cost de la implementació s'ha considerat que tot el treball dut a terme l'ha realitzat un programador. Aquestes hores inclouen la part de documentació, la implementació dels dos plugins, l'execució de les proves i la preparació de l'entorn d'execució. La taula 5.1 mostra el cost d'implementació:

Perfil	**Hores**	**Preu/Hora**	**Total**
Programador	500	25€	12500€

Taula 5.1: Cost d'implementació del projecte

Cost de l'equipament

Per a calcular el cost de l'equipament, es considerarà únicament l'ús de MareNostrum. Com que MareNostrum no s'ha utilitzat en exclusiva, es calcula a partir de les hores de cálcul utilitzades, com si es tractés de qualsevol usuari. A la taula 5.2 es motra el cost total de l'ús de l'equipament.

Concepte	**Hores**	**Preu/Hora**	**Total**
MareNostrum	6500	0.30€	1950€

Taula 5.2: Cost de l'equipament utilitzat

Cost Total

Finalment, a la taula 5.3 es mostra el cost total del projecte.

Concepte	**Cost**
Cost Implementació	12500€
Cost Equipament	1950€
Total	14450€

Taula 5.3: Cost total del projecte

CAPÍTOL 6

Conclusions

L'objectiu d'aquest projecte era implementar i analitzar el comportament de la política de selecció de recursos Less Consume en un entorn de supercomputació real i comparar els resultats obtinguts amb els de les simulacions existents.

Es pot afirmar, doncs, que s'han assolit els objectius marcats a l'inici del projecte. El resultat ha estat la implementació de la política Less Consume com a *plugin* de selecció de recursos d'SLURM mitjançant l'adaptació del *plugin*, ja existent, Consumable Resources. Per tal de poder realitzar l'anàlisi s'ha implementat una modificació del *plugin* d'*Accounting* que permet obtenir la informació dels treballs executats en el format estàndard SWF.

Les proves realitzades han demostrat el correcte funcionament de la política i han validat els resultats obtinguts en les simulacions. S'ha observat com la política Less Consume és capaç de controlar l'assignació del recurs compartit d'ample de banda de memòria. També s'ha comprovat com es produeix un gran increment del temps d'espera dels treballs respecte a Consumble Resources, tot i que hi ha una millora sustancial de la penalització en el temps d'execució d'un treball. Al mateix temps redueix o elimina els treballs matats respecte a la política Consumable Resources, el què provoca que la utilització útil de cpu, tot i no ser molt bona, sigui molt millor.

També s'ha pogut comprovar com les modificacions de la configuració del *plugin* responen al comportament esperat. Tot i així, per tal de poder fer el pas a producció d'aquesta política de selecció de recursos, caldria que els usuaris disposessin de les dades necessàries d'ample de banda de memòria i caldria fer un estudi de com afectaria al sistema un ús incorrecte d'aquesta informació per part dels usuaris. S'hauria d'avaluar el rendiment de la política en un sistema

amb més de 10000 cpus i caldria replantejar-se la metodologia utilitzada per a comptabilitzar les hores consumides per cada treball.

Deixant de banda l'aspecte més pràctic, amb aquest projecte s'han abordat diverses àrees de la implementació de software.S'ha hagut d'estudiar un codi desenvolupat per tercers, entendre la lògica d'un nou algoritme de selecció de recursos i realitzar-ne la implementació en un llenguatge molt utilitzat com és el C. La realització de les proves també ha comportat l'aprenentatge de l'ús de l'aplicació Paraver. Tot plegat ha fet que aquest projecte hagi resultat molt enriquidor a nivell personal.

APÈNDIX A

Workloads generats

A.1 HIGH

```
High:   80
Medium: 10
Low:    10
File:   workload.txt
HIGH: 44
MEDIUM: 3
LOW: 3
   18   00:00:43    4          mg.C.4     3    88
   46   00:04:27    2    sintetichigh    16  1935
  233   00:06:08    4    sintetichigh    22  1935
  271   01:21:02    8          cg.D.8    34  1600
  322   00:01:23    1    sintetichigh     5  1935
  369   00:02:47    2    sintetichigh    10  1935
  573   00:01:23    4    sintetichigh     5  1935
  620   00:05:23    4          cg.C.4    98   800
  668   00:02:53    4          mg.C.4    12    88
  855   00:15:24    8          mg.C.8   528   440
 1094   00:05:34    1    sintetichigh    20  1935
 1160   00:05:51    1    sintetichigh    21  1935
 1222   00:10:19    1    sintetichigh    37  1935
 1227   01:34:18    1    sintetichigh   338  1935
 1269   00:37:23    4    sintetichigh   134  1935
 1372   00:00:00    8          cg.D.8     0  1660
 1401   00:47:59    4    sintetichigh   172  1935
 1506   00:21:52   16         cg.D.16    16  1500
 1644   00:09:32    8          cg.D.8     4  1600
 1749   00:03:54    4    sintetichigh    14  1935
 1877   01:56:47    8          cg.D.8    49  1660
 1952   00:09:34   16         cg.D.16     7  1500
 2042   00:06:25    1    sintetichigh    23  1935
 2087   00:00:33    2    sintetichigh     2  1935
 2124   01:47:15    8          cg.D.8    45  1660
 2199   01:18:39    8          cg.D.8    33  1600
 2340   01:27:53    4    sintetichigh   315  1935
 2614   00:08:05    1    sintetichigh    29  1935
 2648   01:24:42    4          cg.C.4  1540   800
 2810   00:02:23    8          cg.D.8     1  1600
 2978   00:06:06   16         cg.D.16     3  1500
 3171   00:03:37    1    sintetichigh    13  1935
 3217   00:28:44    4    sintetichigh   103  1935
 3323   00:21:27    8          cg.D.8     9  1660
 3387   00:01:23    2    sintetichigh     5  1935
 3429   00:04:11    4    sintetichigh    15  1935
 3528   01:53:16    1    sintetichigh   406  1935
 3727   00:22:52    1    sintetichigh    82  1935
 3776   00:07:06   32         cg.D.32    17  1440
 3911   00:21:02    1    sintetichigh    50  1935
 3989   00:07:48    1    sintetichigh    28  1935
 4128   00:00:00    8          cg.D.8     0  1600
 4488   01:14:46    1    sintetichigh   268  1935
 4643   00:07:48    4    sintetichigh    28  1935
 4707   00:04:46    8          cg.D.8     2  1660
 5070   00:05:01   32         cg.D.32    12  1440
 5116   01:13:53    8          cg.D.8    31  1600
 5182   00:37:23    4    sintetichigh   134  1935
 5301   01:01:30   16         cg.D.16    45  1500
 5395   00:06:19    4          cg.C.4   115   800
```

A.2 MEDIUM

```
High:   50
Medium: 10
Low:    40
File:   workload.txt
HIGH: 28
MEDIUM: 4
LOW: 18
   18   00:00:50    4          cg.B.4    50   580
   46   00:04:27    2    sintetichigh    16  1935
  233   00:06:08    4    sintetichigh    22  1935
  271   01:21:02    8          cg.D.8    34  1660
  322   00:01:23    1    sintetichigh     5  1935
  369   00:02:47    2    sintetichigh    10  1935
  573   00:01:23    4    sintetichigh     5  1935
  620   00:05:25    4          cg.B.4   325   580
  668   00:02:55    4          cg.B.4   175   580
  855   00:14:18    8          cg.D.8     6  1660
 1094   00:05:34    1    sintetichigh    20  1935
 1160   00:05:51    1    sintetichigh    21  1935
 1222   00:10:19    1    sintetichigh    37  1935
 1227   01:34:28    1     sinteticlow   836    83
 1269   00:37:23    4    sintetichigh   134  1935
 1372   00:00:00    8          cg.D.8     0  1660
 1401   00:48:00    4          cg.B.4  2880   580
 1506   00:21:52   16         cg.D.16    16  1500
 1644   00:11:39    8          cg.B.8  1296   445
 1749   00:04:10    4          cg.B.4   250   580
 1877   01:58:11    8          cg.B.8 13133   445
 1952   00:09:34   16         cg.D.16     7  1500
 2042   00:06:25    1    sintetichigh    23  1935
 2087   00:00:33    2    sintetichigh     2  1935
 2124   01:47:28    8          cg.B.8 11942   445
 2199   01:18:38    8          cg.B.8  8738   445
 2340   01:27:53    4    sintetichigh   315  1935
 2614   00:08:05    1    sintetichigh    29  1935
 2648   01:24:42    4          cg.C.4  1540   800
 2810   00:02:23    8          cg.D.8     1  1600
 2978   00:04:09   16         sp.C.16   289   480
 3171   00:03:43    1     sinteticlow    33    83
 3217   00:28:45    4          cg.B.4  1725   580
 3323   00:21:27    8          cg.D.8     9  1600
 3387   00:01:23    2    sintetichigh     5  1935
 3429   00:04:16    4          cg.B.4   256   580
 3528   01:53:16    1    sintetichigh   406  1935
 3727   00:22:49    1     sinteticlow   202    83
 3776   00:07:06   32         cg.D.32    17  1440
 3911   00:14:07    1     sinteticlow   125    83
 3989   00:07:52    1     sinteticmed    40  1050
 4128   00:00:00    8          cg.D.8     0  1600
 4488   01:15:01    1     sinteticlow   664    83
 4643   00:07:55    4          cg.B.4   475   580
 4707   00:04:46    8          cg.D.8     2  1600
 5070   00:05:01   32         cg.D.32    12  1440
 5116   01:13:53    8          cg.D.8    31  1600
 5182   00:37:27    4          cg.C.4   681   800
 5301   00:00:00   16         cg.C.16   681   800
 5395   00:06:08    4    sintetichigh    22  1935
```

A.3 LOW

```
High:   10
Medium: 10
Low:    80
File:   workload.txt
HIGH: 5
MEDIUM: 2
LOW: 43
   18   00:00:50    4          cg.B.4    50   580
   46   00:04:33    2          cg.B.2   167   570
  233   00:06:15    4          cg.B.4   375   580
  271   01:22:54    8          cg.B.8  9212   445
  322   00:01:34    1     sinteticmed     8  1050
  369   00:02:53    2          cg.B.2   106   570
  573   00:01:40    4          cg.B.4   100   580
  620   00:05:25    4          cg.B.4   325   580
  668   00:02:55    4          cg.B.4   175   580
  855   00:15:24    8          cg.B.8  1712   445
 1094   00:05:45    1     sinteticlow    51    83
 1160   00:05:52    1     sinteticlow    52    83
 1222   00:10:23    1     sinteticlow    92    83
 1227   01:34:28    1     sinteticlow   836    83
 1269   00:37:30    4          cg.B.4  2250   580
 1372   00:02:04    8          cg.B.8   231   445
 1401   00:48:00    4          cg.B.4  2880   580
 1506   00:22:10   16         sp.C.16  1538   480
 1644   00:11:39    8          cg.B.8  1296   445
 1749   00:04:10    4          cg.B.4   250   580
 1877   01:58:11    8          cg.B.8 13133   445
 1952   00:09:34   16         sp.C.16   664   480
 2042   00:06:33    1     sinteticlow    58    83
 2087   00:00:49    2          cg.B.2    30   570
 2124   01:47:28    8          cg.B.8 11942   445
 2199   01:18:38    8          cg.B.8  8738   445
 2340   01:28:02    4          cg.B.4  5282   580
 2614   00:08:14    1     sinteticlow    73    83
 2648   01:24:44    4          cg.B.4  5084   580
 2810   00:04:09    8          cg.B.8   462   445
 2978   00:04:09   16         sp.C.16   289   480
 3171   00:03:44    1     sinteticmed    19  1050
 3217   00:28:45    4          cg.B.4  1725   580
 3323   00:22:04    8          cg.B.8  2453   445
 3387   00:01:38    2          cg.B.2    60   570
 3429   00:04:16    4          cg.B.4   256   580
 3528   01:53:13    1     sinteticlow  1002    83
 3727   00:22:49    1     sinteticlow   202    83
 3776   00:07:06   32         cg.D.32    17  1440
 3911   00:13:56    1    sintetichigh    50  1935
 3989   00:07:54    1     sinteticlow    70    83
 4128   00:00:00    8          cg.D.8     0  1600
 4488   01:15:01    1     sinteticlow   664    83
 4643   00:07:55    4          cg.B.4   475   580
 4707   00:06:39    8          cg.B.8   740   445
 5070   00:05:01   32         cg.D.32    12  1440
 5116   01:16:09    8          cg.B.8  8462   445
 5182   00:37:23    4    sintetichigh   134  1935
 5301   01:01:53   16         sp.C.16  4293   480
 5395   00:06:21    4          cg.B.4   381   580
```

APÈNDIX B

Codi benchmark propi

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
#include "/gpfs/apps/CEPBATOOLS/mpitrace/64/include/mpitrace_user_events
    .h"
#include <math.h>
#include <sched.h>

#define SIZE 256000000
#define STRIDE 64

#ifdef NITER
 #define ITERS NITER
#else
 #define ITERS 60
#endif

int main(int argc, char **argv ){

    unsigned int index, disp,iter,rank;
    int *data;
    int niter;
    void **p;
    unsigned long mask ;

    MPI_Init( &argc, &argv );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    data = malloc(SIZE * sizeof(int) + 1);

/*      access the vector:
        We divided the vector in chunks of <stride> entries each.
        We access the first element of each chunk, then the second of
            each chunk, etc.
        We repeat this process ITERS times
        1.)
        1.1) x[0*stride+0], x[1*stride+0], x[2*stride+0], ...
        1.2) x[0*stride+1], x[1*stride+1], x[2*stride+1], ...
```

```
        ...
        1.n) x[0*stride+stride-1], x[1*stride+stride-1], x[2*stride+
            stride-1], ...
        x[stride-1],                x[2*stride-1],                  x[3*
            stride-1\
        ], ...
        2.) IDEM
        ...
        LOOP\_ITERATION.) IDEM
*/

    p = data;
    //Inicialitzacio del traceig
    //MPItrace\_init();
    for(iter=0;iter<ITERS;iter++){
    disp = 0;
    //Inici de la iteracio
    //MPItrace\_eventandcounters (1000, 1);
    while (1) {
   for (index = 0; index <= ( SIZE - STRIDE); index += STRIDE) {
        p = &(data[index + disp]);
        p = *p;
   }
   if (++disp == STRIDE) {
        p = data;
             break;
   }
   printf("Iter:%d\n",disp);
    }
    //Fi iteracio
    //MPItrace\_eventandcounters (1000, 0);
    }
    //Finalitzacio traceig
    //MPItrace\_fini();
    MPI_Finalize();
    return 0;
}
```

Bibliografia

[BSC] *BSC*, http://www.bsc.es.

[CCF+99] Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, James Patton Jones, Scott T. Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby, *Benchmarks and standards for the evaluation of parallel job schedulers*, JSSPP, 1999, pp. 67–90.

[Gui] Francesc Guim, *Job-guided scheduling strategies for multi-site hpc infrastructures*, Ph.D. thesis.

[LF03] Uri Lublin and Dror G. Feitelson, *The workload on parallel supercomputers: modeling the characteristics of rigid jobs*, J. Parallel Distrib. Comput. **63** (2003), no. 11, 1105–1122.

[Par] *Paraver*, http://www.bsc.es/plantillaA.php?cat_id=485.

[SLU] *Simple linux utility for resource management*, http://computing.llnl.gov/linux/slurm/.

[TOP] *Top500 Supercomputer Sites*, http://www.top500.org.